ИНФОРМАТИКА

2016 **апрель-июнь** **№ 2**

ПРИКЛАДНЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

УДК 519.876.5

А.И. Хобня, В.Д. Левчук, О.М. Демиденко

КОНЦЕПТУАЛЬНАЯ МОДЕЛЬ МЕХАНИЗМОВ ОБЕСПЕЧЕНИЯ КАЧЕСТВА ОБСЛУЖИВАНИЯ В СЕТЯХ С КОММУТАЦИЕЙ ПАКЕТОВ

Анализируются и обобщаются принципы работы механизмов обеспечения качества обслуживания в сетях с коммутацией пакетов. Проводится декомпозиция и выделяются четыре компонента механизмов обеспечения качества обслуживания. Описываются особенности каждого из компонентов. Предлагается способ формализации для построения имитационных моделей механизмов обеспечения качества обслуживания в сетях нового поколения, при котором каждый из компонентов представляется функцией от определенных параметров.

Введение

Формализация работы механизмов обеспечения качества обслуживания (Quality of Service, QoS) в сетях с коммутацией пакетов в виде достаточно точных аналитических моделей в настоящее время является неразрешимой задачей. Альтернативным подходом к моделированию сложных систем является имитационное моделирование, представляющее собой такой метод исследования, при котором изучаемая система заменяется ее моделью, с достаточной точностью описывающей реальную систему. С этой моделью и проводятся эксперименты с целью исследования реальной системы.

Для моделирования событий прибытия пакетов разработана имитационная модель, симулирующая генерацию трафика различных типов [1], которые требуют различных показателей качества обслуживания. Как правило, обеспечение различных параметров качества для различных типов трафика осуществляется путем применения различных алгоритмов планирования и управления очередью к пакетам трафика различных классов. Для построения имитационной модели механизмов обеспечения качества обслуживания необходимо осуществить формализацию: провести анализ объекта моделирования, выделить важные для задач моделирования компоненты и определить основные активности с точки зрения систем массового обслуживания.

Целями моделирования работы механизмов обеспечения качества обслуживания в сетях могут выступать:

– оптимизация настроек механизма обеспечения качества обслуживания для достижения лучших показателей качества работы сети;

– вычисление значений показателей качества обслуживания при использовании различных настроек с целью выяснения резервов для увеличения нагрузки сети;

– определение степени влияния различных классов трафика на работу сети;

– определение пороговой нагрузки [2], при которой качество обслуживания будет обеспечено при текущих настройках;

– исследование существующих и разработка новых механизмов обеспечения качества обслуживания.

Популярные системы моделирования сетей (например, NS-2, NS-3) позволяют разработчику модели сети использовать некоторые реализованные в системе имитации модели компонентов обеспечения качества обслуживания (например, некоторые алгоритмы активного управления очередью пакетов).

В отличие от подходов, принятых в существующих популярных системах моделирования сетей, представленный в работе способ позволяет разработчику и исследователю имитационной модели самостоятельно описывать алгоритмы обеспечения качества обслуживания с использованием языков высокого уровня абстракции независимо от деталей реализации системы

имитационного моделирования. Это особенно важно для тестирования и исследования новых разработок в механизмах обеспечения качества обслуживания, реализация которых отсутствует на данный момент в популярных системах моделирования сетей.

В данной работе представлены анализ, обобщение, формализация и способ имитационного моделирования механизмов обеспечения качества обслуживания в сетях передачи данных с коммутацией пакетов. Представленная обобщенная формальная модель может быть использована для построения конкретных имитационных моделей, а также платформы моделирования и исследования механизмов обеспечения качества обслуживания.

1. Обзор механизмов обеспечения качества обслуживания

Рассмотрим механизмы обеспечения, используемые в сетевом оборудовании. Один из крупнейших производителей коммуникационного оборудования разделяет инструменты обеспечения качества обслуживания на средства управления перегрузкой (Congestion Management) и средства предотвращения перегрузки (Congestion Avoidance). К средствам управления перегрузкой относятся FIFO (First Input First Output) Queueing, PQ (Priority Queueing), CQ (Custom Queueing), WFQ (Weighted Fair Queuing), CBWFQ (Class-Based Weighted Fair Queuing) и LLQ (Low Latency Queueing). Средства предотвращения перегрузки включают в себя WRED (Weighted Random Early Detection), DWRED (Distributed Weighted Random Early Detection), Flow-Based WRED и DiffServ Compliant WRED. FIFO Queueing обеспечивает базовые функции хранения и пересылки пакетов. Это простейший алгоритм обработки очереди, который может использоваться по умолчанию в некоторых случаях и не требует конфигурации. PQ позволяет задавать строгие приоритеты для важного трафика и гарантирует первоочередную обработку наиболее важного трафика. Приоритеты для пакетов могут быть гибко настроены в зависимости от портов, адресов источника и приемника, размера пакетов и т. д. CQ резервирует заданную долю общей полосы пропускания сетевого интерфейса для каждого заданного типа трафика. Если какой-либо тип трафика не использует выделенную для него часть полосы пропускания, то ее могут применять остальные типы трафика. WFQ разделяет трафик на несколько потоков, основываясь на таких параметрах, как адрес источника, адрес приемника, порты отправителя и получателя и т. д. К каждому потоку трафика применяется приоритет (или вес). CBWFQ расширяет функциональность WFQ, предоставляя возможность настраивать классы трафика. CBWFQ позволяет выделить точную долю полосы пропускания для каждого класса. Может быть сконфигурировано до 64 различных классов трафика. LLQ позволяет отдать одному из классов трафика абсолютный приоритет и настроить доли полосы пропускания для остальных классов, как и в CBWFQ. WRED является реализацией известного алгоритма RED, который использует значение IP Precedence для обеспечения преимущества трафика с большим приоритетом. DWRED представляет собой расширение WRED и позволяет задавать минимальные и максимальные пороговые значения длины очереди пакетов для трафиков различных классов. Flow-Based WRED и DiffServ Compliant WRED также являются расширениями алгоритма WRED и позволяют разделить управление уничтожением пакетов для различных потоков и различных классов трафика соответственно [3].

Иные производители сетевого оборудования и программного обеспечения применяют похожие концепции для построения механизмов обеспечения качества обслуживания. Также используется концепция Hierarchy Token Bucket для уничтожения или обеспечения приоритетов пакетов [4]. Как правило, механизм качества обслуживания каким-либо образом классифицирует входящие сетевые пакеты, распределяет их по нескольким хранилищам либо уничтожает, затем определяет очередность отправки пакетов из различных хранилищ.

2. Анализ и обобщение механизмов обеспечения качества обслуживания в сетях с коммутацией пакетов

Проанализировав алгоритмы работы механизмов обеспечения качества обслуживания, можно выделить четыре основных компонента каждого механизма обеспечения качества обслуживания: очереди пакетов, алгоритмы классификации пакетов, алгоритмы активного управления очередью и алгоритмы планирования.

Очереди пакетов представляют собой простейшие очереди с последовательной обработкой. Механизм обеспечения качества обслуживания может использовать одну или несколько очередей пакетов. Как правило, различные механизмы отличаются тремя следующими компонентами.

Алгоритмы классификации пакетов обеспечивают определение класса трафика пакета и распределение пакетов различных классов по различным очередям. Существуют классификации по IP Precedence (ToS), по DSCP, основанные на потоках, по MPLS QoS и Deep Packet Inspection (DPI).

Особенности алгоритмов классификации трафика заключаются в использовании:

– специальных полей заголовков IP-пакетов, маркирующих трафик (ToS, DSCP, MPLS QoS);

– IP-адреса отправителя, IP-адреса получателя, порта отправителя и порта получателя;

– информации о прикладных протоколах и приложениях.

Данные свойства необходимо учитывать при построении имитационных моделей механизмов качества обслуживания.

Следующим важнейшим компонентом механизмов обслуживания являются *алгоритмы активного управления очередью* (Active Queue Management, AQM). AQM – это класс алгоритмов, управляющих уничтожением пакетов сетевого интерфейса в случае переполнения или близкого к переполнению состояния внутреннего буфера устройства в целях сокращения нагрузки на сеть. Существуют следующие AQM, используемые в механизмах обеспечения качества обслуживания: Tail Drop, Random Early Detection (RED), Weighted Random Early Detection (WRED), Adaptive Random Early Detection (ARED), Robust Random Early Detection (RRED) [5], RED with Preferential Dropping (RED-PD), Random Exponential Marking (REM), Blue, Stochastic Fair Blue (SFB), Resilient Stochastic Fair Blue (RSFB), Controlled Delay (Codel), Active Virtual Queue (AVQ) [6], PI Controller.

Проанализировав данные алгоритмы, отметим их особенности:

– возможность настройки (большинство алгоритмов имеют настраиваемые параметры);

– управление одной очередью;

– использование данных о длине очереди, объеме буфера и его применении;

– использование данных об источнике, приемнике и других полях пакета (например, WRED идентифицирует трафик различных классов, SFB поддерживает несколько потоков трафика и т. д.);

– хранение и изменение внутреннего состояния (например, Blue хранит и изменяет параметр вероятности, RED-PD хранит историю уничтожения пакетов и т. д.);

– возможность уничтожения пакета большинством алгоритмов до помещения его в очередь (однако, например, CoDel уничтожает пакеты при удалении их из очереди).

Эти свойства необходимо учитывать при построении имитационных моделей механизмов обеспечения качества обслуживания. Также следует иметь в виду, что если механизм обеспечения качества обслуживания использует несколько очередей пакетов, то для различных очередей могут применяться различные алгоритмы активного управления.

Еще одним важнейшим компонентом механизмов обеспечения качества обслуживания являются *алгоритмы планирования*. Алгоритмы данного класса управляют последовательностью отправляемых пакетов. Существуют следующие алгоритмы планирования отправки пакетов, используемые в механизмах обеспечения качества обслуживания: First in First Out (FIFO), Priority Queuing (PQ), Round-robin, Weighted Round-robin (WRR), Weighted Fair Queuing (WFQ), Low Latency Queuing (LLQ).

Для алгоритмов планирования характерны:

– возможность настройки (большинство алгоритмов имеют настраиваемые параметры);

– использование данных о состоянии всех очередей;

– хранение и изменение внутреннего состояния (например, WFQ хранит и изменяет параметры виртуального времени прибытия и отправки пакетов).

3. Формализация и построение имитационных моделей механизмов обеспечения качества обслуживания

С точки зрения теории сетей массового обслуживания механизм обеспечения качества обслуживания может быть представлен компонентом, накапливающим заявки перед устройством обслуживания, которым является исходящий интерфейс данного сетевого узла. Пакеты представляются в качестве заявок на обслуживание (или транзактов). Каждый транзакт должен содержать информацию, необходимую для моделирования работы механизма обеспечения качества обслуживания и для вычисления показателей качества обслуживания: виртуальное модельное время отправки пакета, порт отправителя, порт получателя, IP-адреса отправителя и получателя, размер пакета, используемый протокол прикладного уровня, тип передаваемых данных, значения полей ToS, DSCP, MPLS QoS, дополнительную информацию.

Для моделирования работы механизма обеспечения качества обслуживания необходимо описать его поведение при наступлении двух дискретных событий: прибыл новый входящий пакет, следующий исходящий пакет может быть отправлен.

Используя приведенный выше анализ, поведение механизма обеспечения при прибытии нового пакета можно представить в виде UML-диаграммы активностей (рис. 1). Также поведение может быть описано в псевдокоде следующим образом:

```
// Поведение механизма QoS при прибытии нового пакета
номер очереди = Классифицировать пакет (данные_пакета);
пакет должен быть уничтожен = Применить AQM (данные_пакета, номер очереди);
Если пакет должен быть уничтожен:
    уничтожить пакет;
Иначе:
    поместить пакет в очередь (номер очереди);
```

Вначале осуществляется классификация пакета для определения очереди, в которую он может быть помещен. Затем применяется механизм AMQ для выбранной очереди с целью определения, должен ли данный пакет быть уничтожен. После этого пакет помещается в очередь или отбрасывается.

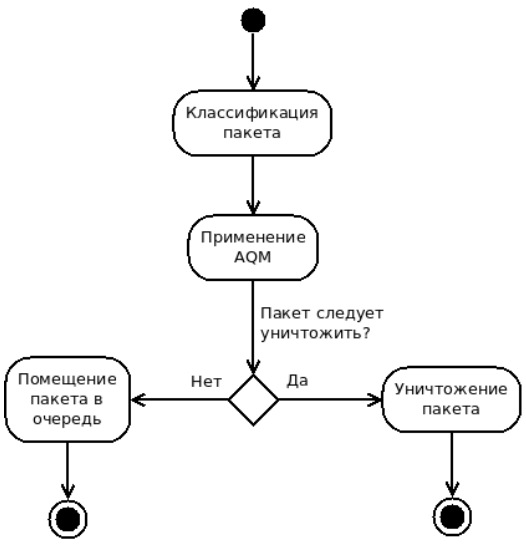

Рис. 1. Диаграмма активностей поведения механизма качества обслуживания при прибытии нового пакета

Используя приведенный выше анализ, поведение механизма обеспечения качества обслуживания при наступлении события, когда следующий исходящий пакет может быть отправлен, можно представить в виде UML-диаграммы активностей (рис. 2). Также поведение может быть описано в псевдокоде следующим образом:

```
// Поведение механизма QoS при отправке пакета
номер очереди = Вызвать алгоритм планирования(данные_очередей);
данные пакета = извлечь пакет из очереди (номер очереди);
пакет должен быть уничтожен = Применить AQM (данные_пакета, номер очереди);
Если пакет должен быть уничтожен:
    уничтожить пакет;
Иначе:
    отправить пакет;
```

Вначале выполняется работа алгоритма планирования для определения очереди, из которой должен быть отправлен следующий пакет. Затем применяется вторая часть алгоритма активного управления очередью для определения, должен ли быть уничтожен отправляемый пакет. Пакет удаляется из очереди и после этого отправляется или уничтожается.

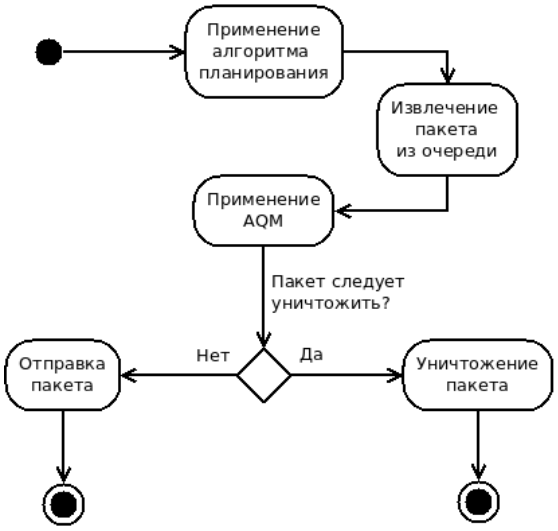

Рис. 2. Диаграмма активностей поведения механизма качества обслуживания при готовности отправки следующего пакета

Алгоритм классификации может быть задан некоторой функцией $f_c(P, S_c)$, где P – кортеж значений параметров пакета, а S_c – текущее состояние памяти модуля классификации, которая может принимать значения из множества всех возможных пар (n, S_c'), где n – номер очереди, в которую необходимо поместить пакет, а S_c' – новое состояние памяти модуля классификации. Алгоритмы активного управления очередями могут быть заданы функциями $f_a(P, q, S)$, где P – кортеж значений параметров пакета, q – кортеж значений состояния очереди, а S – текущее состояние памяти модуля активного управления очередями, которые могут принимать значения из множества всех возможных пар (r, S'), где r – результат работы, который принимает значения из булева множества, а S' – новое состояние памяти модуля AMQ. Алгоритм планирования может быть задан функцией $f_s(Q, S_s)$, где Q – кортеж значений состояния очередей, а S_s – текущее состояние памяти модуля планирования, которая может принимать значения из множества всех возможных пар (n, S_s'), где n – номер очереди, из которой необходимо отправить пакет, а S_s' – новое состояние памяти модуля планирования. Таким образом, механизм обслуживания может быть задан параметрами

$$n,\ f_c(P, S_c),\ f_{ai1}(P, q_1, S_1), ..., f_{ain}(P, q_n, S_n),\ f_{ao1}(P, q_1, S_1), ..., f_{aon}(P, q_n, S_n),\ f_s(Q, S_s),$$

где n – количество очередей; $f_c(P, S_c)$ – функция алгоритма классификации; $f_{ai1}(P, q_1, S_1), ..., f_{ain}(P, q_n, S_n)$ – функции алгоритмов AMQ всех очередей от 1 до n, применяемых для входящих пакетов; $f_{ao1}(P, q_1, S_1), ..., f_{aon}(P, q_n, S_n)$ – функции алгоритмов AMQ всех очередей от 1 до n, применяемых для исходящих пакетов; $f_s(Q, S_s)$ – функция алгоритма планирования.

С помощью концепций объектно-ориентированного программирования описанные выше функции могут быть представлены классами, реализующими соответствующие интерфейсы с одним методом. Данный подход использован при построении разрабатываемого проблемно-ориентированного инструментария автоматизации имитационного моделирования сетей нового поколения. Функции алгоритмов классификации представляются классами, реализующими интерфейс Classification, который содержит единственный метод classify(Packet packet), возвращающий номер очереди, в которую необходимо поместить пакет. Функции алгоритмов активного управления очередями представляются классами, реализующими интерфейс ActiveQueueManagement, который содержит методы shouldBeDropedByArrival(Packet packet, Queue queue) и shouldBeDropedBySending(Packet packet, Queue queue), возвращающие значения из булева множества. Функции алгоритмов планирования представляются классами, реализующими интерфейс Scheduling, который содержит единственный метод schedule(List<Queue> queue), возвращающий номер очереди, из которой необходимо отправить пакет. Инструментарий содержит реализации для моделирования множества существующих алгоритмов механизмов обеспечения качества обслуживания и позволяет пользователю задавать произвольные алгоритмы, используя языки программирования Java, JavaScript, Jython, JRuby, Scala, Groovy, Clojure и некоторые другие.

4. Пример использования рассматриваемого способа формализации для построения имитационной модели механизма обеспечения качества обслуживания

В качестве примера рассмотрим задачу моделирования механизма, использующего четыре очереди для трафика с различными приоритетами, которые определяются на основе меток MPLS, и вариант алгоритма ARED для предотвращения перегрузки в данных очередях.

Допустим, для пакетов с полем MPLS TC (traffic class), равным 6 или 7, необходимо использовать очередь 3, для пакетов с полем *TC*, равным 4 или 5, – очередь 2 и т. д., т. е. номер очереди вычисляется как [*TC*/2], где *TC* – значение поля MPLS TC (traffic class).

Как указано выше, алгоритм классификации может быть задан некоторой функцией $f_c(P, S_c)$, где P – кортеж значений параметров пакета, а S_c – текущее состояние памяти модуля классификации, которая может принимать значения из множества всех возможных пар (n, S_c'), где n – номер очереди, в которую необходимо поместить пакет. В данном случае алгоритм классификации не сохраняет никакого состояния, поэтому S_c' и S_c всегда являются пустыми кортежами. Так как в этом примере используются четыре очереди, n принимает значения из множества {0, 1, 2, 3}. Кортеж параметров пакета содержит различные характеристики, такие как время прибытия, размер и т. д. Для данного алгоритма классификации важно только наличие значения поля MPLS TC, которое принимает целые значения из интервала [0; 7]. Как указано выше, с помощью концепций объектно-ориентированного программирования функции алгоритмов классификации представляются классами, реализующими интерфейс Classification, который содержит единственный метод classify(Packet packet), возвращающий номер очереди, в которую необходимо поместить пакет. Например, реализация описанного выше алгоритма классификации, использующего исключительно поле MPLS QoS TC, может быть задана на языке Jython следующим образом:

```
# Реализация алгоритма классификации на основе поля MPLS QoS TC
class MplsQosClassification (Classification):

  # Функция классификации пакета
  def classify(self, packet):
    return packet.mplsTrafficClass / 2
```

Рассмотрим алгоритм активного управления очередями в данном примере. Для каждой очереди используется вариант алгоритма ARED. Как отмечено выше, алгоритмы активного управления очередями могут быть заданы функциями $f_a(P, q, S)$, где P – кортеж значений параметров пакета, q – кортеж значений состояния очереди, а S – текущее состояние памяти модуля активного управления очередями, которые могут принимать значения из множества всех воз-

можных пар (r, S'), где r – результат работы, который принимает значения из булева множества, а S' – новое состояние памяти модуля AMQ. В данном примере S и S' состоят из значений следующих величин: *avg* – средний размер буфера очереди, *size* – текущий размер буфера очереди, p – текущая вероятность уничтожения пакета и *last_time* – последнее время выполнения. Кортеж q не используется в этом случае, а P содержит различные характеристики пакета, такие как время прибытия, размер и т. д. Для алгоритма активного управления очередью важно только наличие значений следующих полей: *size* – размер пакета, *arrival_time* – время прибытия пакета, *current_time* – текущее время. Как указано выше, функции алгоритмов активного управления очередями могут быть представлены классами, реализующими интерфейс ActiveQueueManagement, который содержит методы shouldBeDropedByArrival(Packet packet, Queue queue) и shouldBeDropedBySending(Packet packet, Queue queue), возвращающие значения из булева множества. Например, реализация модификации алгоритма активного управления очередью ARED может быть задана на языке Jython следующим образом:

```
# Реализация алгоритма активного управления очередью ARED
class AredManagement (ActiveQueueManagement):
  # Инициализация
  def __init__(self, min_threshold, max_threshold):
   # Шаг инкремента
   self.alpha = 0.01
   # Шаг уменьшения
   self.beta = 0.9
   # Целевой размер очереди
   self.target = min_threshold + 0.5*(max_threshold - min_threshold)
   # Начальное состояние
   self.avg = 0
   self.size = 0
   self.p = 0.01
   self.last_time = 0

  # Функция принятия решения об уничтожении или сохранении пакета
  def shouldBeDropedByArrival(self, packet, queue):
    # Обновить средний размер очереди
    self.avg = (self.avg*self.last_time + packet.size *
        (packet.arrival_time - self.last_time))/ packet.arrival_time
    # Обновить вероятность уничтожения пакета
    if(self.avg > self.target and self.p <= 0.5):
      self.p += self.alpha*(packet.arrival_time - self.last_time)
    elif self.avg < self.target and self.p >= 0.01:
      self.p *= self.beta**(packet.arrival_time - self.last_time)
    # Обновить последнее время
    self.last_time = packet.arrival_time
    # Пакет следует уничтожить?
    decision = random()
    if decision < self.p:
      # Уничтожить пакет
      return True
    else
      # Положить пакет в очередь
      self.size += packet.size
      return False

  # Функция принятия решения об уничтожении или сохранении пакета
  def shouldBeDropedBySending(self, packet, queue):
   # Обновить средний размер очереди
   self.avg = (self.avg*self.last_time + packet.size *
        (packet.current_time - self.last_time))/ packet.current_time
   # Обновить вероятность уничтожения пакета
```

```
    if(self.avg > self.target and self.p <= 0.5):
      self.p += self.alpha*(packet.current_time - self.last_time)
    elif (self.avg < self.target and self.p >= 0.0.1):
      self.p *= self.beta**(packet.current_time - self.last_time)
    # Обновить последнее время
    self.last_time = packet.current_time
    # обновить текущий размер очереди
    self.size -= packet.size
    return False
```

Рассмотрим алгоритм планирования в данном примере. Как указано выше, алгоритм планирования может быть задан функцией $f_s(Q, S_s)$, где Q – кортеж значений состояния очередей, а S_s – текущее состояние памяти модуля планирования, которая может принимать значения из множества всех возможных пар $(n, S_s^{'})$, где n – номер очереди, из которой необходимо отправить пакет, в $S_s^{'}$ – новое состояние памяти модуля планирования. Поскольку в примере используются четыре очереди, n принимает значения из множества $\{0, 1, 2, 3\}$. Кортежи $S_s^{'}$ и $S_s^{'}$ содержат значения параметров *virStart* и *virFinish* для последнего пакета каждой очереди. Параметр *virStart* представляет собой виртуальное время прибытия пакета из вершины очереди, параметр *virFinish* – виртуальное время отправки. Кортеж Q содержит кортежи значений параметров пакетов P из вершины каждой очереди. Кортежи P содержат различные характеристики пакета, такие как время прибытия, размер и т. д. Для данного алгоритма планирования важно только наличие значений следующих полей: *size* – размер пакета, *arrival_time* – время прибытия пакета. Как указано выше, функция алгоритма планирования может быть представлена классом, реализующим интерфейс Scheduling, который содержит единственный метод schedule (List<Queue> queue), возвращающий номер очереди. Например, реализация алгоритма планирования WFQ может быть задана на языке Jython следующим образом:

```
# Реализация алгоритма планирования WFQ
class WeightedFairQueueing (Scheduling):
  # Инициализация
  def __init__(self, weights):
    self.weights = weights
    queues_number = len(weights)
    self.number = queues_number
    # Начальное состояние
    self.virStart = [0]*queues_number;
    self.virFinish = [0]*queues_number;

  # Функция планирования
  def schedule(self, queues):
    # Обновить сумму весов активных потоков
    active_weight = 0
    for i in xrange(0, len(queues)):
      if not queues[i].isEmpty():
        active_weight += self.weight[i]

    # Обновить виртуальное время начала и конца для новых пакетов
    for i in xrange(0, len(queues)):
      if not queues[i].isEmpty() and self.virStart[i] < 0:
        packet = queues[i].peek()
        self.virStart[i] = max(packet.arrival_time,
                                        self.virFinish[i])
        self.virFinish[i] = (virStart[i] +
                        packet.size/(self.weights[i]/active_weight))

    # Выбрать очередь
    minVirFinish = self.virFinish[0]
    for i in xrange(0, len(queues)):
      queue = queues[i]
```

```
        if not queue.isEmpty() and self.virFinish[i] < minVirFinish )
          minVirFinish = self.virFinish[i]
          queueNum = i

      # Использовать очередь
      self.virStart[queueNum] = -1
      self.virFinsih[queueNum] = -1
      return queueNum
```

Система моделирования вызывает заданные пользователем алгоритмы в последовательностях, представленных диаграммами активностей на рис. 1 и 2, при наступлении соответствующих событий. Для моделирования событий прибытия пакетов система имитирует генерацию трафика различных типов [1].

Предложенный подход позволяет описывать алгоритмы обеспечения качества обслуживания с использованием языков высокого уровня, абстрагируясь от деталей реализации системы имитационного моделирования. Декомпозиция механизмов обеспечения качества обслуживания на компоненты четырех типов позволяет создавать имитационные модели новых механизмов, комбинируя существующие в системе компоненты и (или) переопределяя некоторые из них.

Заключение

В данной работе представлены анализ, обобщение и способ формализации механизмов обеспечения качества обслуживания в сетях передачи данных с коммутацией пакетов. Предложенный способ использован для разработки инструментария автоматизации имитационного моделирования сетей нового поколения, который позволяет моделировать и исследовать механизмы обеспечения качества обслуживания.

Список литературы

1. Демиденко, О.М. Концептуальная модель генерации VoIP трафика в сети NGN / О.М. Демиденко, А.И. Хобня // Известия Гомельского гос. ун-та им. Ф. Скорины. – 2014. – № 6(87). – С. 117–122.

2. Кулинченко, В.Н. Об одном подходе к определению пропускной способности каналов ЛВС по протоколам TCP/IP, ICMP и UDP / В.Н. Кулинченко, О.М. Демиденко, П.Л. Чечет // Проблемы физики, математики и техники. – 2014. – № 4(21). – С. 1–3.

3. Cisco IOS Quality of Service Solutions Configuration Guide [Electronic resource]. – 2015. – Mode of access : http://www.cisco.com/c/en/us/td/docs/ios/12_2/qos/configuration/guide/fqos_c/qcfintro.html. – Date of access : 01.03.2015.

4. Traffic Control HOWTO [Electronic resource]. – 2015. – Mode of access : http://www.tldp.org/HOWTO/Traffic-Control-HOWTO/components.html. – Date of access : 01.03.2015.

5. RRED: Robust RED algorithm to counter low-rate denial-of-service attacks / C. Zhang [et al.] // IEEE Communications Letters. – 2010. – no 14(5). – P. 489–491.

6. Kunniyur, S.S. An adaptive virtual queue (AVQ) algorithm for active queue management / S.S. Kunniyur, R. Srikant // Networking, IEEE/ACM Transactions on. – 2004. – Vol. 12, no. 2. – P. 286–299.

Поступила 03.02.2016

Гомельский государственный университет
им. Франциска Скорины,
Гомель, Советская, 104
e-mail: akhobnya@ya.ru

A.I. Khobnia, V.D. Liauchuk, O.M. Demidenko

A CONCEPTUAL MODEL OF QUALITY OF SERVICE MECHANISMS IN PACKET-SWITCHED NETWORKS

The paper presents analysis and generalization of the quality of service mechanisms in packet-switched networks. Decomposition of the quality of service mechanisms is presented. Each QoS control mechanism can be split into four components. Each component is responsible for the defined subtask. A method of formalization is offered in which each QoS mechanism component is presented by function of defined arguments. Presented method is used for simulation of QoS mechanisms in the next generation networks.