Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования

**«Уральский федеральный университет**

**Имени первого Президента России Б.Н. Ельцина»**

**Институт математики и компьютерных наук**

**Кафедра алгебры и дискретной математики**

**Автоматизация процесса тестирования коммуникационного оборудования на основе генетических алгоритмов**

|  |  |
| --- | --- |
| Допустить к защите: | Дипломная работа  студента 6 курса  Перевощикова И.В.  Научный руководитель**:**  Доцент кафедры алгебры и дискретной математики ИМКН УрФУ,  кандидат технических наук  Синадский Н.И. |

Екатеринбург

2015

# РЕФЕРАТ

Перевощиков И.В. Автоматизация процесса тестирования коммуникационного оборудования на основе генетических алгоритмов, дипломная работа: **TODO**

Ключевые слова: СИНТЕЗ ТРАФИКА, ГЕНЕТИЧЕСКИЙ АЛГОРИТМ, ТЕСТИРОВАНИЕ СЕТЕВОГО ОБОРУДОВАНИЯ

Объект исследования – эволюционные подходы к процессу тестирования коммуникационного оборудования на предмет выявления пороговых значений характеристик оборудования.

Цель работы – разработка и реализация программного комплекса на базе генетического алгоритма для автоматического тестирования сетевого оборудования.

В процессе работы проводилось исследование работы генетических алгоритмов и их применения в тестировании коммуникационного оборудования.

В результате работы был разработан программный код, реализующий комплексное тестирование сетевого оборудования с использованием автоматически сгенерированного трафика, близкого по своим характеристикам к реальному.

Результаты работы могут применяться разработчиками сетевого оборудования для автоматического тестирования на предмет выявления условий, являющихся для данного оборудования экстремальными.

# Содержание

Оглавление

[РЕФЕРАТ 2](#_Toc410311213)

[Содержание 3](#_Toc410311214)

[Введение 4](#_Toc410311215)

[Глава 0. Протоколы передачи данных 7](#_Toc410311216)

[Глава 1. Используемое программное обеспечение и библиотеки 15](#_Toc410311217)

[Глава 2. Модель сети 19](#_Toc410311218)

[Глава 3. Генетический алгоритм 30](#_Toc410311219)

[Заключение 34](#_Toc410311220)

[Список использованной литературы 35](#_Toc410311221)

[Приложение 36](#_Toc410311222)

# Введение

Целью данной работы является автоматизация исследования коммуникационного оборудования на выявление таких конфигураций сети, которые будут являться для тестируемого сетевого оборудования «экстремальными».

Другими словами, требуется разработать программный комплекс, автоматически оптимизирующий конфигурацию сети, в которой установлено тестируемое оборудование, таким образом, чтобы привести некоторые значимые для исследования характеристики данного оборудования к «крайним» (пороговым) значениям. Так как поле возможных вариантов сетей охватить методами грубой силы не представляется возможным из-за временных ограничений, было решено воспользоваться средствами генетического алгоритма.

Тема генетических алгоритмов исследована довольно широко, и генетические алгоритмы нашли свое применение в таких областях науки как

* Экстремальные задачи (нахождение точек минимума и максимума),
* Задачи о кратчайшем пути,
* Задачи компоновки,
* Составление расписаний,
* Аппроксимация функций,
* Отбор (фильтрация) входных данных,
* Настройка искусственной нейронной сети,
* Моделирование искусственной жизни (Artificial life systems),
* Биоинформатика (свертывание белков и РНК),
* Игровые стратегии,
* Нелинейная фильтрация,
* Развивающиеся агенты/машины (Evolvable agents/machines).

Данная работа является попыткой применить генетический алгоритм в автоматизации тестирования сетевого оборудования. С этой целью был разработан программный комплекс, состоящий из трех частей:

1. Реализация математической модель конфигурации сети, в которую входят подсети, узлы в этих подсетях и «потоки» между узлами модели, а также принимающие участие в эволюции функции распределения вероятности принятия какой либо характеристикой потока какого-либо значения из заданного отрезка.
2. Непосредственная реализация генетического алгоритма, который производит формирование популяции, скрещивание и отбор. «Особью» в данном алгоритме является некоторая конфигурация сети, представленная при помощи математической модели.
3. Модуль, отвечающий за тестирование некоторой конфигурации сети, полученной из генетического алгоритма, возвращающий для каждой конфигурации некоторую оценку, необходимую для произведения дальнейшего отбора генетическим алгоритмом.

Общая схема взаимодействия модулей представлена на рисунке 1.



Рис. . Общая схема работы программного комплекса

В качестве языка программирования в данной работе был выбран Python, как язык, наиболее эффективный при реализации сетевых взаимодействий с достаточной степенью гибкости настройки последних.

В качестве экспериментального объекта исследований – типичный представитель семейства «домашних» маршрутизаторов – маршрутизатор производства компании D-Link, модель DIR-300.

# Глава 0. Протоколы передачи данных

*Сетевой протокол* — набор правил и действий (последовательность действий), позволяющий осуществлять соединение и обмен данными между двумя и более включёнными в сеть устройствами.

Сетевые протоколы описывают инструкции взаимодействия для устройств, которые подключены к сети. Строятся такие протоколы чаще всего по многоуровневому принципу. Протокол некоторого уровня определяет одно из технических правил связи. В настоящее наиболее распространенная модель для классификации сетевых протоколов - модель OSI (Open System Interconnection — взаимодействие открытых систем, ВОС).

*Модель OSI* — это иерархическая модель работы сети, включающая в себя несколько уровней. Модель OSI представлена набором протоколов и правил связи, организованных в 7 уровней:

1. *Физический уровень* получает пакеты данных от вышележащего канального уровня и преобразует их в оптические или электрические сигналы, соответствующие 0 и 1 бинарного потока, а также производит обратное преобразование. Эти сигналы посылаются через среду передачи на приемный узел.
2. *Канальный уровень* обеспечивает создание, передачу и прием кадров данных. Этот уровень обслуживает запросы сетевого уровня и использует сервис физического уровня для приема и передачи пакетов.
3. *Сетевой уровень* отвечает за деление пользователей на группы. На этом уровне происходит маршрутизация пакетов на основе преобразования MAC-адресов в сетевые адреса. Сетевой уровень обеспечивает также прозрачную передачу пакетов на транспортный уровень.
4. *Транспортный уровень* делит потоки информации на достаточно малые фрагменты (пакеты) для передачи их на сетевой уровень.
5. *Сеансовый уровень* отвечает за организацию сеансов обмена данными между оконечными машинами.
6. *Уровень представления* отвечает за возможность диалога между приложениями на разных машинах. Этот уровень обеспечивает преобразование данных (кодирование, компрессия и т.п.) прикладного уровня в поток информации для транспортного уровня.
7. *Прикладной уровень* отвечает за доступ приложений в сеть. Задачами этого уровня является перенос файлов, обмен почтовыми сообщениями, управление сетью и т.п.

Разработанный программный комплекс формирует заголовки пакетов 3 и 4 уровней, поскольку именно на этих уровнях модели OSI работает большая часть коммутационного оборудования, в частности – маршрутизаторы. Представители 3 уровня это IP и ICMP протоколы, а 4 уровня — UDP и TCP протоколы.

3.1 TCP протокол

Протокол TCP (Протокол управления передачей) – один из основных протоколов для передачи данных по сети, описан в RFC 793. Данный протокол используется в случаях, когда необходима надежная, гарантированная передача данных к узлу назначения. Например, при загрузке веб-страниц.

Формат TCP пакета приведен на рисунке \*.



Рисунок \*. Формат TCP - пакета

Заголовок дейтаграммы состоит из 32-разрядных слов и имеет переменную длину, зависящую от размера поля «Опции», но всегда кратную 32 битам. За заголовком непосредственно следуют данные, передаваемые в датаграмме.

Поле «Флаги» содержит 6 битовых флагов:

1. URG — Поле «Указатель важности» задействовано (Urgent pointer field is significant),
2. ACK — Поле «Номер подтверждения» задействовано (Acknowledgement field is significant),
3. PSH — инструктирует получателя отправить данные, накопившиеся в приемном буфере, в приложение пользователя,
4. RST — Оборвать соединение, очистить буфер (Reset the connection),
5. SYN — Синхронизация номеров последовательности,
6. FIN — флаг указывает на завершение соединения.

Также важно отметить, что для гарантированной доставки пакета, необходимо знать IP-адреса источника и назначения. Для того, чтобы сохранить в рамках требований модели OSI реализацию протокола, на стороне отправителя генерируется псевдозаголовок, содержащий необходимые данные, с учетом которого вычисляется контрольная сумма, но сам псевдозаголовок не отправляется, а генерируется заново на узле назначения для проверки контрольной суммы.

3.2 UDP протокол

Протокол датаграмм пользователя UDP (User Datagram Protocol) разработан Дэвидом П. Ридом в 1980 году и описан в документе RFC 768 «User Datagram Protocol (UDP)». UDP используется некоторыми программами вместо TCP для быстрой, простой, но ненадежной передачи данных между узлами TCP/IP. К примеру, для передачи потокового видео или голосовых вызовов, то есть в условиях, где важнее скорость передачи, нежели высокая степень целостности передаваемой информации.

UDP обеспечивает службу датаграмм, не ориентированную на установление соединения, что означает, что UDP не гарантирует ни доставку, ни правильность порядка доставки датаграмм. Узел-источник, которому требуется надежная связь, должен использовать либо протокол TCP, либо программу, которая сама обеспечивает подтверждения и следит за правильностью порядка датаграмм.

На рисунке \* показан формат UDP – пакета.



Рисунок \*. Структура UDP - пакета.

Заголовок UDP состоит из четырёх полей, каждое по 2 байта (16 бит). Порт источника и контрольная сумма необязательны к использованию в IPv4, в то время как в IPv6 необязателен только порт отправителя.

В поле *«Порт отправителя»* указывается номер порта отправителя. Предполагается, что это значение задаёт порт, на который при необходимости будет посылаться ответ. В противном же случае, значение должно быть равным 0. В поле *«Порт получателя»* содержится номер порта получателя, данное поле является обязательным. В поле *«Длина пакета»* указывается длина всей датаграммы, включая IP и UDP заголовки.

3.3 ICMP протокол

Протокол межсетевых управляющих сообщений ICMP (Internet Control Message Protocol) является обязательным стандартом TCP/IP, описанным в документе RFC 792, «Internet Control Message Protocol (ICMP)». Используя ICMP, узлы и маршрутизаторы, связывающиеся по протоколу IP, могут сообщать об ошибках и обмениваться ограниченной управляющей информацией и сведениями о состоянии.

ICMP-сообщения обычно автоматически отправляются в следующих случаях.

* IP-датаграмма не может достичь узла назначения.
* IP-маршрутизатор (шлюз) не может перенаправлять датаграммы с текущей скоростью передачи.
* IP-маршрутизатор перенаправляет узел-отправитель на другой, более выгодный маршрут к узлу назначения.

На рисунке \* показан формат ICMP – пакета.



Рисунок **\***. Формат ICMP - пакета

На основании полей «Тип» и «Код» определяется тип ICMP – сообщения. Содержимое пакета может включать в себя различные данные, в зависимости от типа и кода пакета, например, заголовки пакета, который послужил причиной ошибки и отправки данного ICMP – сообщения. Наиболее распространенные типы ICMP – сообщений приведены в таблице \*.

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Код | Сообщение | Описание |
| 0 | 0 | Эхо-ответ | Отвечает на эхо-запрос ICMP. |
| 8 | 0 | Эхо-запрос | Определяет, доступен ли в сети IP-узел. |
|  | 1 | Узел недоступен | Информирует узел о том, что датаграмма не может быть доставлена. |
| 4 | 0 | Замедление источника | Требует от узла снизить скорость отправки датаграмм, так как в сети возник затор. |
| 5 |  | Перенаправление | Информирует узел о наличии лучшего маршрута. |
| 11 |  | Истечение времени | Сообщает, что время жизни IP-датаграммы истекло. |

Таблица \*. Распространенные типы ICMP – сообщений.

В разработанном программном комплексе генерируются ICMP-сообщения всех возможных типов, с разной вероятностью возникновения сообщения того или иного типа, для обеспечения генерации сетевых пакетов, по общему распределению параметров близкого к распределению этих параметров реальной сети.

# Глава 1. Используемое программное обеспечение и библиотеки

Разработанный программный комплекс может быть развернут на компьютере под управлением операционной системы семейства Linux, с установленным Python 2.7, библиотеками Pyevolve и Scapy.

Разработка велась на компьютере с операционной системой Ubuntu 14.04 LTS, с установленными Python 2.7.6, Pyevolve 0.6rc и Scapy 2.1.1.

1.1 Библиотека Pyevolve

На данный момент существует множество способов реализации генетических алгоритмов. Самым быстрым и удобным способом для языка Python является библиотека Pyevolve [4].

Данная библиотека был разработан специально для создания полноценных генетических алгоритмов. Сама библиотека – лишь каркас, реализующий общую логику работы генетического алгоритма, но все же сильно упрощающий жизнь разработчика.

Для быстрого создания работающего генетического алгоритма авторы библиотеки предлагают реализовать только функциональную составляющую генетического алгоритма, как, например, программное представление особи или алгоритм скрещивания.

Всю общую и неизменную логику работы генетического алгоритма выполняет сам библиотека: генерирует популяции, производит отбор, скрещивания, мутации, проверяет критерий остановки эволюции и т.п.

Pyevolve также предоставляет возможности логирования, выгрузки данных в различные типы хранилищ (база данных, электронная таблица), взаимодействия с текущей популяцией в интерактивном режиме.

Простота реализации генетического алгоритма хорошо видна из нижеследующего примера, в котором реализован алгоритм поиска двоичного вектора длины 20 с наибольшим количеством нулей.

1. **def** eval\_func(chromosome):
2. score = 0.0
3. **for** value **in** chromosome:
4. **if** value==0:
5. score += 1
6. **return** score
8. genome = G1DList.G1DList(20)
9. genome.evaluator.set(eval\_func)
10. ga = GSimpleGA.GSimpleGA(genome)
11. ga.evolve(freq\_stats=10)
12. **print** ga.bestIndividual()

Листинг 1. Пример простейшего ГА

В строках 1-6 описывается функция, ставящая в соответствие каждой особи некое число или оценку «приспособленности». В строках 8-12 производится несложная настройка параметров работы генетического алгоритма.

Таким образом, библиотека Pyevolve предоставляет разработчику возможность сконцентрироваться на своей конкретной задаче, предоставляя удобный интерфейс для быстрой реализации требуемого генетического алгоритма и возлагая на себя всю «сервисную» работу.

1.2 Утилита Scapy

Scapy – утилита, написанная на языке Python, позволяющая пользователю посылать, просматривать, анализировать и генерировать сетевые пакеты.  В отличие от аналогичных утилит, утилита Scapy не ограничена только теми протоколами, пакеты которых она может генерировать. Фактически, она позволяет создавать любые сетевые пакеты, что обеспечивает требуемую для решения поставленной задачи гибкость.

Типичные примеры использования утилиты:

* Сканирование,
* Трассировка маршрута,
* Юнит-тестирование всевозможных сетевых функций,
* Различные виды сетевых атак.

К примеру, классический ICMP-ping с выводом всех ответивших на запрос узлов, с помощью Scapy реализуется всего в две строки:

1. ans,unans=sr(IP(dst="192.168.1.1-254")/ICMP())
2. ans.summary(**lambda** (s,r): r.sprintf("%IP.src% is alive") )

Листинг 2. Пример реализации сетевого взаимодействия

В данной работе утилита Scapy использовалась для следующих целей:

1. Создание TCP-пакетов,
2. Создание UDP-пакетов,
3. Создание ICMP-пакетов,
4. Создание файла-выгрузки в формате pcap, содержащего сгенерированные моделью сети пакеты.

Ниже приведены примеры использования утилиты Scapy в программном коде данной работы. Так, например, создается TCP – пакет:

1. l3 = IP(src=ip1, dst=ip2)  # создание IP-пакета
2. l4 = TCP(sport=self.port1, dport=self.port2)  # создание TCP-
3. пакета
4. ...
5. l34 = l3 / l4 # Инкапсуляция TCP-пакета в IP-пакет
6. l5 = self.generate\_l5(params['flp'].random())  # генерирование
7. полезной нагрузки
8. ...
9. l34[TCP].flags |= flags\_on  # установка флагов TCP
10. l34[TCP].flags &= ~flags\_off
11. l34[TCP].seq = seq
12. l34[TCP].ack = ack
13. l34[IP].ttl = params['fttl'].random()  # установка времени жизни
14. пакета
15. p = l34 / l5  # помещение полезной нагрузки в готовый пакет

Листинг 3. Пример создания TCP - пакета

Аналогичным образом создается UDP – пакет:

1. l3 = IP(src=ip1, dst=ip2)
2. l4 = UDP(sport=self.port1, dport=self.port2)  # создание UDP –
3. пакета
4. l34 = l3 / l4
5. l5 = self.generate\_l5(params['flp'].random())
6. l34[IP].ttl = params['fttl'].random()
7. p = l34 / l5
8. p.time = t   # установка времени отправки пакета

Листинг 4. Пример создания UDP - пакета

Создание ICMP – пакета происходит аналогично вышеприведенным примерам. А вот так происходит сохранение набора сетевых пакетов:

1. wrpcap("temp.cap",pkts)  # pkts – предварительно сгенерированный
2. массив пакетов

Листинг 5.Сохранение массива пакетов

Таким образом, утилита Scapy, является наиболее подходящим инструментом, для создания сетевых пакетов, параметры которых генерируются в процессе работы генетического алгоритма, и полностью избавляет от необходимости самостоятельной реализации модуля создания сетевого пакета с многочисленными варьируемыми параметрами.

# Глава 2. Модель сети

Так как целью данной работы является тестирование коммутационного оборудования, конечным результатом работы атомарной итерации генетического алгоритма должен быть некий набор сетевых пакетов, которые можно отправить на один из интерфейсов устройства и принять на другом.

Следовательно, вопрос состоит в том, каким именно образом генерировать такие последовательности пакетов. Общих подходов к решению задачи генерации пакетов два:

1. Создать модель генерации, основываясь на том, как выглядят обрабатываемые пакеты с точки зрения самого тестируемого устройства;
2. Создать процесс генерации на основе моделирования сети, имеющей возможность существовать в реальности и «подключить» к этой сети тестируемое устройство.

Для реализации разработанного программного комплекса был выбран второй вариант, поскольку он обладает рядом преимуществ, по сравнению с первым.

Во-первых, при реализации второго варианта, можно надеяться на помещение тестируемого устройства в условия, близкие к реальным условиям эксплуатации, ведь то, что устройство покажет негативные результаты тестов, которые не могут существовать в реальности, слабо может претендовать на результат качественного тестирования.

Во-вторых, основываясь на первом подходе, будет довольно сложно получить последовательность пакетов, в реальности являющуюся, например, неизвестным до сих пор вариантом атакующего воздействия. Во втором варианте такая ситуация является более вероятной.

Таким образом, одним из ключевых моментов данной работы является задача выявления таких параметров сети, которые будут изменяться в процессе работы генетического алгоритма, сохраняя при этом приближенность к параметрам реальных сетей. А также вопрос программного представления такой модели сети.

Итак, в качестве изменяемых параметров для моделирования сети были предложены:

Количество подсетей в моделируемой сети, а также типы этих сетей и их «положение» относительно тестируемого оборудования. То есть, некоторые из подсетей имеют доступ к устройству через один интерфейс (назовем его *«правым»*), а некоторые – через другой (назовем его *«левым»*).

Подсети, в зависимости от интерфейса, через который они имеют доступ к тестируемому устройству, будем называть соответственно *«левыми»* и *«правыми»*. Для наглядности, одна из возможных моделей изображена на рисунке \*.



Рисунок \*. Принципиальная схема расположения подсетей относительно устройства.

В качестве изменяемых параметров подсети были выбраны тип сети и ее расположение относительно тестируемого устройства («левая» или «правая»).

Для узлов моделируемой сети параметрами оказались принадлежность к какой-либо подсети, а также набор сетевых взаимодействий, или потоков, в которых участвует данный узел.

*Потоком* в данной работе назовем набор сетевых пакетов одного сетевого протокола, отправляемых между двумя узлами сети в обоих направлениях. Другими словами, в данной модели сети поток описывает некое сетевое взаимодействие, например, передачу файла, если протоколом данного потока является TCP, или, к примеру, голосовой вызов, если протоколом данного потока оказался UDP.

У потока предложенных изменяемых параметров гораздо больше:

* Длительность самого потока по времени,
* Направление отправки пакета (от первого узла ко второму, либо от второго к первому),
* Временя жизни сетевого пакета,
* Размер полезной нагрузки пакета,
* Время отправки пакета.

Нетрудно заметить, что значения большинства этих параметров распределены в реально существующих сетях неравномерно, вследствие чего в модель необходимо ввести для каждого, требующего этого изменяемого параметра, функцию распределения вероятности случайной величины.

2.1 Функция распределения вероятностей случайной величины

*Случайной* называют величину (далее - СВ), которая в результате испытания примет одно и только одно возможное значение, наперед не известное и зависящее от случайных причин, которые заранее не могут быть учтены [6].

*Дискретной* называют СВ, которая принимает отдельные, изолированные воз­можные значения с определенными вероятностями [6].

Следует отметить, что в рамках данной работы все СВ считаются дискретными, с конечным числом возможных значений.

*Законом распределения* дискретной случайной величины называют соответствие между возможными значениями и их вероятностями [6]. Такой закон можно задать, например, в табличной форме.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *X* | *x1* | *x2* | *…* | *xn* |
| *p* | *p1* | *p2* | *…* | *pn* |

Таблица \*. Закон распределения СВ.

Приняв во внимание, что в одном испытании случайная величина принимает· одно и только одно возможное зна­чение, заключаем, что события образуют полную группу, следовательно, сумма вероят­ностей этих событий, т. е. сумма вероятностей второй строки таблицы, равна единице:

(1) [6].

Для того чтобы воспользоваться возможностями получения некоторого случайного значения случайной величины требуется представить ее ФРВ в программном виде. С этой целью был предложен следующий подход. Для каждой ФРВ будем хранить:

* Тип значений данной СВ – целый или вещественный,
* Минимальное значение, которое может принимать данная СВ,
* Максимальное значение, которое может принимать данная СВ,
* Закон распределения данной СВ.

При этом программное представление первых трех пунктов очевидно, но для закона распределения было предложено использовать особую конструкцию для удобства проведения испытаний в дальнейшем. Представим набор пар вида где а и - минимальное и максимальное значение СВ соответственно. При этом Пример такого набора изображен на рисунке \*.

Рисунок \*. Графическое представление моделируемой конструкции.

Из рисунка наглядно видно, как получить результат испытания. Нужно случайным образом бросить точку на ось абсцисс, а затем, найти ближайшее от оси абсцисс пересечение с какой-либо горизонталью вертикальной прямой, проведенной через эту точку. Координата оси ординат полученной точки и будет результатом испытания.

Формально закон распределения случайной величины из такого набора можно получить следующим образом.

1. Отсортировать элементы набора по возрастанию .
2. Из каждой пары соседних элементов отсортированного набора , где получить столбец табличного представления закона распределения случайной величины, где значение вероятности - , значение, принимаемое СВ с такой вероятностью - .
3. Кроме того, к результирующей таблице необходимо добавить значение вероятности с принимаемым значением

Очевидно, что построенный закон полностью удовлетворяет равенству (1), а значит, является законом распределения некоторой случайной величины.

Программная реализация получения результата испытания при такой конструкции закона распределения становится довольно простой, что иллюстрирует листинг \*.

1. points = [[0.2, 42], [1.0, 9]]
2. r = random.random()
3. i = 0
4. **while** r > points[i][0]:
5. i += 1
6. **return** points[i][1]

Листинг \*. Пример получения результата испытания.

В строке 1 задается набор пар, описывающий закон распределения. Важно отметить, что в таком наборе обязательно должен присутствовать элемент с , для удовлетворения ограничениям (равенство (1)). В строках 2 – 5 производится генерация случайного вещественного числа и поиск диапазона, в который это число попадает, с последующим нахождением результата текущего испытания. Важно отметить, что генерируемое число – равномерно распределенная СВ, а значит, вероятности возникновения тех или иных значений СВ, для которой ищется результат испытаний, строго совпадают соответствующим законом распределения.

В результате программное представление СВ является базовым классом FX со следующим набором методов и данных:

* Поле *v\_type*, содержащее в себе тип значений СВ – целый или вещественный,
* Поле *v\_from*, содержащее в себе минимальное возможное значение СВ,
* Поле *v\_to*, содержащее максимально возможное значение, принимаемое СВ.
* Массив *points*, который соответствует описанному выше набору пар.
* Метод *random()*, возвращающий результат независимого испытания для данной СВ.
* Вспомогательные методы для работы генетического алгоритма, о которых будет сказано ниже.

2.2 Поток

Понятие потока в данной работе введено главным образом для того, чтобы стал понятен процесс генерации сетевых пакетов на основе построенной модели. Программным представлением потока стал базовый класс Flow, представляющий собой программный интерфейс и содержащий в себе следующие данные и методы:

* Поля *node1* и *node2*, содержащие в себе номера узлов сети, которые связывает данный поток;
* Поле ftf содержащее в себе объект класса FX, который описывает СВ – время жизни потока;
* Поле fhf содержащее объект класса FX, описывает СВ – направление отправки пакетов;
* Для каждого из двух направлений – набор полей (fttl, flp, ftp) содержащий в себе СВ – время жизни пакета, СВ – длина полезной нагрузки (данных), СВ – время отправки пакета;
* Метод генерации полезной нагрузки generate\_l5(), данный метод принимает на вход один параметр – требуемую длину данных;
* Метод генерации набора пакетов generate(), параметры которых поддаются законам распределения соответствующих случайных величин. Входные параметры данного метода – время начала открытия потока между узлами t0 и объект класса Translator, единственная задача которого – преобразовывать виртуальные узлы модели в корректные ip-адреса.

В рамках разработанного программного комплекса данный интерфейс реализуют три класса – FlowTCP, FlowUDP и FlowICMP, которые соответствуют потокам с сетевыми протоколами ЕМЗ, UDP и ICMP.

Пример реализации ключевого метода generate() для класса FlowICMP представлен в листинге \*. Для двух других классов реализация данного метода весьма схожа с представленной реализацией.

1. **def** generate(self, translator, t0):
2. ip1 = translator.node2ip[self.node1]
3. ip2 = translator.node2ip[self.node2]
4. l3\_1 = IP(src=ip1, dst=ip2)
5. l4\_1 = ICMP(type=self.type1)
6. l3\_2 = IP(src=ip2, dst=ip1)
7. l4\_2 = ICMP(type=self.type2)
8. l34\_1 = l3\_1 / l4\_1
9. l34\_2 = l3\_2 / l4\_2
10. params1 ={'ftp':self.ftp1,'flp':self.flp1,'fttl':self.fttl1}
11. params2 ={'ftp':self.ftp2,'flp':self.flp2,'fttl':self.fttl2}
12. seq = 0
13. ack = 0
14. packets = []
15. t1 = t0 + self.ftf.random()
16. t = t0
17. **while** t < t1:
18. **if** **not** self.fhf.random():
19. l34 = l34\_1
20. params = params1
21. l34['ICMP'].seq = seq
22. ack = seq
23. seq += 1
24. **else**:
25. l34 = l34\_2
26. params = params2
27. l34['ICMP'].ack = ack
29. tp = params['ftp'].random()
31. l5 = self.generate\_l5(params['flp'].random())
32. l34['IP'].time = t
33. l34['IP'].ttl = params['fttl'].random()
34. p = l34 / l5
35. packets.append(p)
36. t += tp
37. **return** packets

Листинг \*. Пример генерирования набора пакетов.

Из данного примера ясно видно, что ключевым моментом генерации пакетов является получение того или иного параметра пакета путем вызова метода random(), который отвечает за результат независимого испытания.

2.3. Подсети и узлы

Для полноценного моделирования работы сети, к которой «подключено» тестируемое устройство, требуется моделировать не только взаимодействия между узлами сети, но также каким-то образом представлять сами узлы, а также подсети, в которых эти узлы расположены.

Для каждой подсети в рамках данной работы были введены два параметра: тип сети и ее положение относительно тестируемого устройства.

Было выделено несколько типов подсетей: обычные сети, принадлежащие какому-либо классу (A, B, C, D или E), также особые вымышленные подсети, необходимые для того, чтобы обозначить некоторые сетевые явления, например, тип сети «multicast», который требуется для моделирования потока связанного с широковещательным запросом.

Программное представление каждой подсети достаточно тривиально: пара вида (тип, положение), к примеру, листинг \* демонстрирует задание параметров для двух сетей. Первая сеть класса A получает доступ к тестируемому устройству через «левый» интерфейс, вторая сеть – класса B «подключена» к устройству через «правый» интерфейс.

1. nets = [('a', 'l'), ('b', 'r')]

Листинг \*. Пример задания параметров подсетей.

Для узлов в моделируемой сети значимых параметров два: принадлежность к какой-либо сети и потоки, в образовании которых участвует данный сетевой узел. Так как второй параметр уже учтен при моделировании потока, программное представление узлов также становится тривиальной задачей, что демонстрирует листинг \*.

1. nets = [('a', 'l'), ('b', 'r')]
2. nodes = [0, 1, 0]

Листинг \*. Программное представление узлов сети.

Элемент с индексом i в массиве nodes соответствует узлу с номером i создаваемой модели. А значение nodes[i] равняется номеру подсети, к которой данный узел принадлежит. Сети также нумеруются в соответствии с их индексом в массиве nets. В приведенном примере заданы две сети и три узла. Из сказанного выше следует, что в данной модели сети класса A, которая подключена к «левому» интерфейсу, принадлежит два узла с номерами 0 и 2.

Таким образом, была построена программная модель сети, к которой виртуально подключено тестируемое устройство, обладающая множеством характеристик, способных изменяться. Тем самым было создано пространство для работы генетического алгоритма по поиску таких сетевых конфигураций, в которых тестируемое устройство может проявить собственные уязвимости, либо предел своих возможностей.

# Глава 3. Генетический алгоритм

Как было показано выше, при помощи библиотеки Pyevolve, задача программирования генетического алгоритма сводится к представлению модели генома особи в программном виде, что уже было произведено, а также программированию четырех операций: произвольной инициализации генома, мутации генома, скрещивания двух особей и функции вычисления «жизнеспособности» индивидуума.

Геномом особи для работы генетического алгоритма стал класс NetworkGenome, инкапсулирующий в себе все составные части предложенной модели сети. А именно, данный класс содержит в себе набор подсетей, узлов и потоков, которые в совокупности представляют собой моделируемую сеть. Также данный класс содержит в себе требуемые для корректной работы библиотеки Pyevolve методы clone() и copy(), которые возвращают полный клон модели и копируют все свойства одной модели в другую соответственно.

Инициализация произвольной конфигурации сети довольно тривиальна, при выборе подхода «от общего к частному». В рамках разработанного комплекса такие сложные объекты как классы FX и Flow, обладают собственными методами random\_initialize(), каждый из которых возвращает произвольный объект требуемого класса.

Поэтому, основная функция произвольной инициализации конфигурации сети состоит лишь из генерирования случайного числа подсетей, узлов в этих подсетях и потоков между этими узлами. Схема реализации такой функции представлена на рисунке \*.



Рисунок \*. Схема работы функции инициализации объекта класса NetworkGenome.

3.1 Скрещивание.

Задача скрещивания двух геномов в рамках генетического алгоритма в общем случае выглядит как получение нового генома, с признаками обоих входных геномов или «родителей» – «отца» и «матери». В силу особенностей библиотеки Pyevolve, функция скрещивания должна возвращать не одного потомка, а сразу двоих – «брата» и «сестру», что не отменяет накладываемых на потомков ограничений.

Так как, очевидно, важнейшим свойством модели сети с точки зрения тестирования коммутационного оборудования являются потоки, присутствующие в данной модели, при скрещивании был выбран путь обратный последовательности произвольной инициализации и де-факто являющийся одноточечным кроссинговером.

Каждая родительская особь имеет собственный набор потоков, каждый из этих наборов делится на два в случайно выбранном месте, а затем, обмениваются получившимися частями. Общую схему работы кроссинговера иллюстрирует таблица \*.

|  |  |
| --- | --- |
| Родители | Потомки |
| [f1,f2 ...,fn-1,fn,…,fm] | [f1,f2 ...,fn-1,gn,…,gm] |
| [g1,g2 ...,gn-1,gn,…,gm] | [g1,g2 ...,gn-1,fn,…,fm] |

Таблица \*. Наборы потоков родителей и потомков,

fi и gi - некоторые объекты класса Flow

После того, как произошел обмен потоками, потомки наследуют также сети и узлы, которые участвуют в образовании соответствующих потоков. И, таким образом, реализуется операция скрещивания, при которой потомки получают рекомбинированные гены родителей.

3.2 Мутация

Как известно, оператор мутации требуется в генетических алгоритмах для предотвращения схождения эволюции к точке локального экстремума, которая может оказаться не самым оптимальным решением задачи.

В рамках текущей задачи, например, требуется с помощью оператора мутации предотвратить схождение популяции к моделям, просто насыщенным большими объемами передаваемых данных, что по существу есть не что иное, как обыкновенное нагрузочное тестирование.

В целом в смоделированной сети может изменяться одно из трех множеств объектов: множество подсетей, множество узлов или множество потоков. При этом изменение одного множества может повлечь за собой изменение двух других.

Библиотека Pyevolve предоставляет разработчику использовать несколько операторов мутации одновременно, при этом можно выбрать – использовать ли их все, либо использовать каждый раз один произвольный. Наиболее удобным способом явилась реализация трех различных функций мутации для каждого из вышеперечисленных множеств. Логика работы этих операторов довольно схожа: в каждом множестве может мутировать или удалиться существующий элемент, либо добавиться новый. Пример реализации функции мутации множества потоков приведен в листинге \*.

1. def flow\_mutator(genome, \*\*args):
2. choice = random.randint(0, len(genome.flows) + 1)
3. **if** choice < len(genome.flows):
4. genome.flows[choice].mutation()
5. **elif** choice == len(genome.flows):
6. genome.flows.append(
7. random\_flow(random.randint(0,len(genome.nodes)1),
8. random.randint(0, len(genome.nodes) - 1)))
9. **else**:
10. del genome.flows[random.randint(0,len(genome.flows)-1)]
11. **return** 1

Листинг \*. Функция мутации множества потоков.

# Заключение

# Список использованной литературы

1. Панченко Т. В. Генетические алгоритмы [Текст]: учебно-методическое пособие / под ред. Ю. Ю. Тарасевича. — Астрахань: Издательский дом «Астраханский университет», 2007. — 87 с.
2. Лутц М. Изучаем Python, 4-е издание. / Пер. с англ. – Санкт-Петербург: Символ-Плюс, 2011. – 1280 с., ил.
3. Сегаран Т. Программируем коллективный разум. / Пер. с англ. – Санкт-Петербург: Символ-Плюс, 2008. – 368 с., ил.
4. Christian S. Perone. Pyevolve Documentation Release 0.5 [Электронный ресурс] – Режим доступа: http://pyevolve.sourceforge.net/latex\_0\_5.pdf, свободный.
5. Philippe Biondi and the Scapy community. Scapy v2.1.1-dev. documentation [Электронный ресурс] – Режим доступа: <http://www.secdev.org/projects/scapy/doc/>, свободный.
6. Гмурман В. Е. Теория вероятностей и математическая статистика: Учеб. пособие для вузов / 9-е изд., стер. — М.: ВЫСШ. ШХ., 2003. — 479 с., ил.

# Приложение