Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра автоматизации систем вычислительных комплексов

Антипина Анна Вячеславовна

Разработка и исследование метода предотвращения DDoS атак на контроллер в программно-конфигурируемых сетях на основе оценки поведения хостов

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Научный руководитель:
ассистент Пашков В.Н.

Москва, 2020

Аннотация

В работе рассматривается задача предотвращения DDoS атак на контроллер в программно-конфигурируемых сетях (ПКС). По результатам обзора существующих методов предотвращения DDoS атак в ПКС/OpenFlow сетях на контроллер проведен сравнительный анализ методов. В работе реализован собственный метод на основе мониторинга и анализа поведения хостов в сети. Разработанный метод учитывает динамику активности зараженных хостов и изменения в топологии сети.

Метод реализован в виде приложения для ПКС/OpenFlow контроллера RUNOS 2.0 и проведено его экспериментальное исследование.

Содержание

Введение

Программно-конфигурируемые сети (Software-Defined Networking, ПКС) [1] являются новым перспективным направлением в развитии компьютерных сетей. Концепция ПКС подразумевает отделение контура управления сетью от контура передачи данных. Контур данных представляет собой набор коммутаторов (сетевых элементов или ресурсов контура данных), которые связаны между собой каналами связи и содержат минимальный необходимый пакет функций для пересылки, обработки трафика и первоначальной настройки. Контур управления содержит контроллер – программное обеспечение, установленное на сервере, – на котором централизованы функции контроля и управления сетью.

Отношения между контурами строятся на модели клиент-сервер (Рис. 1). Согласно [2] ПКС-контроллер поддерживает такие функциональные интерфейсы, как A-CPI (Application-Controller Plane Interface) – между контуром управления и приложениями, позволяющий реализовывать общие сетевые сервисы в виде приложений для контроллера; D-CPI (Data-Controller Plane Interface) – между контуром данных и контуром управления, предназначенный для управления контроллера ресурсами контура данных.

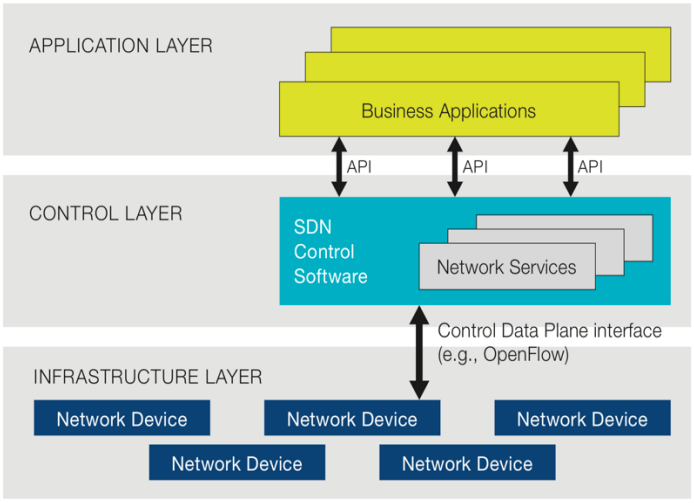

Рис. 1 Архитектура программно-конфигурируемых сетей [2]

Для внедрения данной концепции в реальные сети необходимо в достаточной степени обеспечить безопасное и надежное взаимодействие пользователей сети. Как и в традиционных сетях, важной проблемой безопасности являются *атаки на компьютерные системы* (КС) – это действия, предпринимаемые злоумышленниками и заключающиеся в поиске и использовании уязвимости КС [3].

Среди различных компьютерных атак наиболее распространенными в современном мире являются DoS и DDoS атаки. *DoS атака* (Denial of Service, отказ в обслуживании) представляет собой целенаправленный комплекс действий для частичной или полной остановки работы веб-сайта или другого сетевого ресурса [4]. Атакующий может достичь своей цели разными способами, но в основном все сводится к перегрузке атакуемого ресурса большим количеством запросов. В результате сервер не может с прежней скоростью выполнять свои функции. *DDoS атака* (Distributed DoS) является распределенной DoS атакой и отличается от обычной только тем, что проводится с нескольких компьютерных устройств [5].

Для злоумышленников в ПКС возникают новые возможности вывода сети из строя. Теперь все управляющие функции выполняются на контроллере, а, значит, злоумышленники могут добиться перегрузки контроллера, вследствие чего работоспособность сети будет нарушена. Таким образом, в ПКС появляется новый тип атак – DDoS атаки на контроллер. Она становится возможна благодаря принципу взаимодействия контроллера и коммутаторов, для которого используется протокол взаимодействия.

Наиболее известным протоколом взаимодействия контроллера и коммутаторов в ПКС является протокол OpenFlow [6]. Принцип работы протокола подразумевает существование в коммутаторе нескольких таблиц потоков. Каждая запись в таблице содержит поля соответствия, счетчики и набор инструкций. Все таблицы пронумерованы, начиная с нуля, и поиск соответствия пакетам в таблицах начинается с нулевой. Для первого пакета в потоке выбирается наиболее подходящая по полям соответствия запись в таблице с наибольшим приоритетом и выполняются соответствующие инструкции. Если такой записи не нашлось, то дальнейшие действия определяются конкретной реализацией: пакет может быть сброшен, перенаправлен к следующей таблице или отправлен контроллеру. Инструкции в каждой записи потока содержат действия по обработке пакетов и могут прекращать их обработку и пересылать на выходной порт или продолжать обработку в следующих таблицах потоков, но перенаправляя пакеты только к таблицам с номером больше, чем текущий – пересылки назад запрещены. В случае перенаправления возможна передача информации в виде метаданных и все действия по изменению состояния потока накапливаются в наборе действий потока. Когда встречается инструкция, инициирующая поток выйти из конвейера обработки, весь набор действий применяется к пакетам и они перенаправляется на выходной порт.

Контроллер и коммутатор взаимодействуют по специальному каналу OpenFlow с помощью сообщений. Коммутатор может отправлять на контроллер сообщения без запроса контроллера (асинхронные сообщения). Наиболее трудозатратными с точки зрения потребления физических ресурсов сервера, на котором расположен контроллер, являются асинхронные сообщения Packet-In – запросы на установление нового потока, то есть такого потока, для которого не найдено ни одного подходящего правила в таблице потоков коммутатора, на который поступил данный поток. При поступлении нового потока (Рис. 2), коммутатор буферизирует пакеты потока, формирует и отправляет сообщение Packet-In в контроллер. Получая Packet-In, контроллер рассчитывает новые маршруты, формирует и отправляет управляющие сообщения (Flow-Mod и Packet-Out) в необходимые коммутаторы. После получения сообщений от контроллера, коммутаторы изменяют или добавляют новые правила в соответствующие таблицы.

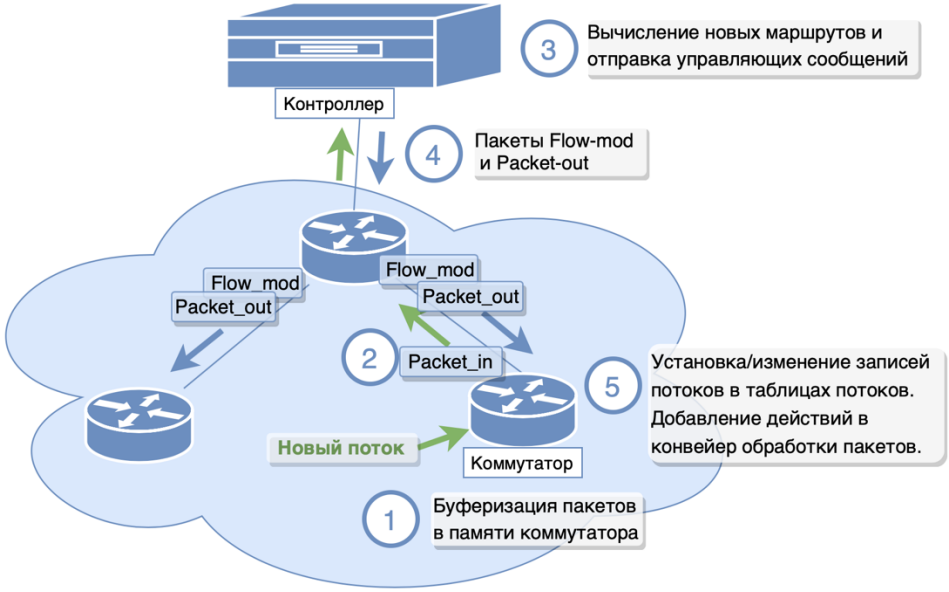

Рис. 2 Алгоритм установления нового потока в таблицы потоков коммутаторов

С увеличением количества таких запросов повышается утилизация ресурсов сервера, на котором расположен контроллер, – заполняется очередь обработки запросов в буфере контроллера, расходуется вычислительная мощность на расчет новых маршрутов, формирование и отправку управляющих сообщений коммутаторам. Поэтому злоумышленники могут добиться перегрузки контроллера, когда он будет загружен обработкой лишь *фиктивных* потоков, то есть потоков, сформированных злоумышленниками, а *пользовательские* – потоки от обычных пользователей – не будут успевать обрабатываться, сбрасываясь по таймауту.

Таким образом, происходит сбой работы сервисов сети, вследствие невозможности доставки трафика между серверами и клиентами. Также последствием DDoS атаки на контроллер является истощение буферной памяти и утилизация свободного места в таблицах потоков коммутаторов, через которые совершалась атака. Поэтому задача своевременного обнаружения атаки и противодействия ей является актуальной важной задачей безопасности ПКС, без решения которой невозможно говорить о полноценном внедрении данной концепции в реальных сетях.

В главах 1 и 2 сформулирована цель работы и постановка задачи в неформальном и формальном видах. В главе 3 производится обзор существующих методов предотвращения DDoS атак в ПКС. Глава 4 описывает разработку метода предотвращения DDoS атак на контроллер в ПКС/OpenFlow сети. Далее в главе 5 представлено описание программной реализация предложенного метода и в главе 6 проводится экспериментальное исследование данного метода.

1 Цель работы

Целью работы является разработка и исследование метода предотвращения DDoS атак с динамически меняющейся активностью зараженных хостов на контроллер в программно-конфигурируемых сетях с изменяющейся топологией.

Для достижения поставленной цели необходимо выполнить следующие ***задачи***:

1. Сформулировать и формализовать задачу предотвращения DDoS атаки на контроллер в ПКС/OpenFlow сети.
2. Проанализировать существующие методы обнаружения и предотвращения DDoS атак на контроллер в ПКС.
3. Разработать метод предотвращения DDoS атаки на контроллер, учитывающий возможности динамического изменения топологии сети и вредоносной активности зараженных хостов DDoS атаки.
4. Реализовать разработанный метод в виде приложения для ПКС контроллера RUNOS 2.0.
5. Разработать методику экспериментального исследования метода.
6. Провести экспериментальное исследование метода и проанализировать полученные результаты.

2 Постановка задачи

2.1 Неформальная постановка

Пусть задана программно-конфигурируемая сеть под управлением ПКС контроллера. Топология сети может изменяться динамически. Управление сетью осуществляется по протоколу OpenFlow не ниже версии 1.3 [6]. В узлах сети расположены коммутаторы, физические порты которых подключены к другим коммутаторам, серверам сетевых сервисов (например, SMTP-сервер, DNS-сервер, HTTP-сервер и т.д.) или хостам. *Граничными коммутаторами* назовем такие коммутаторы, которые имеют непосредственное подключение к хостам. Порты граничных коммутаторов, связанные с другими коммутаторами или серверам , назовем *доверенными*, остальные – *хостовыми.*

Вводятся следующие предположения для сети:

- Коммутаторы сети могут отключаться и подключаться во время работы сети.
- Хост может назначить IP адрес самостоятельно или получить его по запросу от DHCP сервера.
- Хосты могут генерировать любой вид трафика (TCP или UDP).
- Хост может менять точку доступа к сети (смена порта и/или коммутатора).
- Хост может отключаться и подключаться во время работы сети.
- Новые хосты могут подключаться во время работы сети.

Предполагается, что в сети присутствуют *зараженные* хосты, доступные злоумышленнику, доля которых от общего количества хостов может изменяться. Остальные хосты в сети будем называть *пользовательскими*. *Активными* зараженными хостами назовем хосты, участвующие в создании фиктивного трафика во время атаки. *Неактивные* зараженные хосты не проявляют вредоносного поведения во время атаки.

Вводятся следующие предположения относительно злоумышленника:

- Злоумышленник знает, что данная компьютерная сеть является ПКС.
- Злоумышленник может управлять зараженными хостами с помощью установленного на них вредоносного программного обеспечения.
- Злоумышленник может «заражать» пользовательские хосты, удаленно устанавливая на них вредоносное программное обеспечение для совершения

DDoS атаки. Таким образом, пользовательские хосты могут переходить в категорию зараженных.

- Злоумышленник может изменять количество активных хостов во время атаки.
- Зараженные хосты могут переходить в категорию пользовательских, если злоумышленник потерял к ним доступ.
- Зараженные хосты могут генерировать TCP и UDP пакеты со случайными IP и MAC адресами отправителя и получателя.
- Злоумышленник может приостанавливать и возобновлять атаку.
- У злоумышленника нет доступа к коммутаторам сети.

Предполагается, что злоумышленник использует следующую стратегию проведения DDoS атаки на контроллер (Рис. 3) – максимизировать количество запросов на установление новых потоков с зараженных хостов. Добиться этого он может с помощью частой отправки пакетов со случайно сформированными IP и MAC адресами источника и получателя с зараженных хостов. Будем называть потоки, сформированные таким образом *фиктивными*, а остальные потоки – *пользовательскими*. Так как работоспособность контроллера ограничена физическими ресурсами сервера, на котором он расположен, целью злоумышленника является перегрузка контроллера, вследствие чего происходит переполнение очередей входного буфера контроллера, задержка обработки или сброс пользовательских потоков, задержка обработки запросов об изменениях топологии в топологии сети и сбой работы сетевых сервисов для пользователей. Также последствием DDoS атаки является заполнение таблиц потоков коммутаторов правилами для фиктивных потоков;

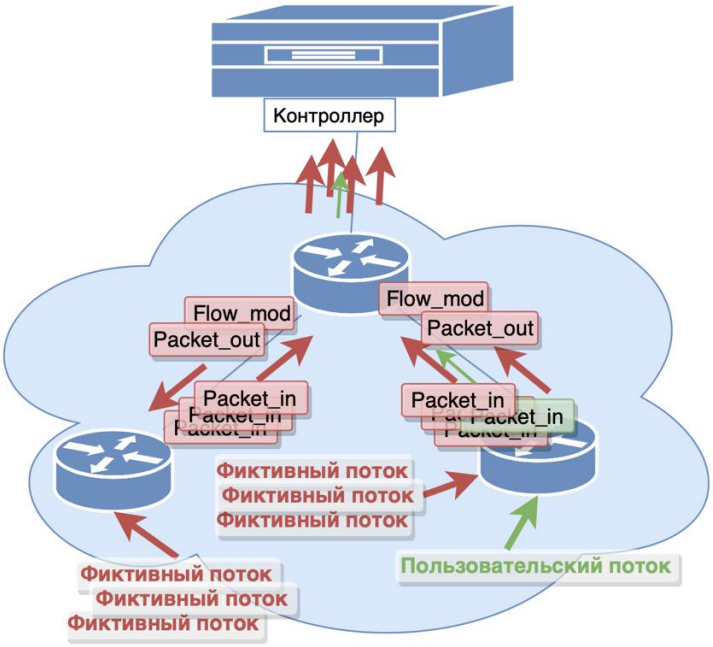

Рис. 3 Алгоритм проведения DDoS атаки на контроллер

Для решения задачи необходимо:

- Определить зараженные хосты.
- Определить действия для предотвращения DDoS атаки с зараженных хостов.
- Определить действия для устранения последствий DDoS атаки.

2.2 Формальная постановка

Пусть задана ПКС/OpenFlow сеть с нефиксированной топологией, состоящая из множества коммутаторов $S = \{S_1 \dots S_{N_S}\}$ и управляемая контроллером C. Пусть коммутаторы из множества $S_E = \{S_1 \dots S_{N_E}\}, S_E \subseteq S$ являются граничными. Пусть к сети подключено множество хостов $H = \{h_1 \dots h_{N_H}\}$ и сетевые сервисы $Serv = \{Serv_1 \dots Serv_{N_{Serv}}\}$, IP адреса которых известны контроллеру. Среди сервисов сети присутствует DHCP сервер в виде приложения, установленного на контроллере.

Динамика изменения топологии обуславливается следующими возможностями поведения хостов и коммутаторов во время работы сети:

- Хост может менять точку подключения к сети (порт коммутатора, коммутатор).
- Хост h_i ($h_i \in H$) может быть отключен ($H = H \backslash \{h_i\}, N_h = N_h - 1$).

- Хост h_j ($h_j \notin H$) может быть подключен $\left(H = H \cup \{h_j\}, N_h = N_h + 1\right)$.
- Коммутатор S_k ($S_k \in S$) может быть отключен $(S = S \backslash \{S_k\}, N_S = N_S - 1)$ и подключен вновь во время работы сети.
- Смена IP адреса хоста может быть произведена хостом статически.
- Смена IP адреса хоста может быть произведена по протоколу DHCP.

В сети присутствует злоумышленник, пользующийся хостом h_A (Рис. 4). Злоумышленник имеет доступ к зараженным хостам $H^A = \{h_1 \dots h_{N_A}\}, H^A \subset H$, остальные хосты считаются пользовательскими $H^U, H^U = H \backslash H^A$. Хосты из множества $H^{AA} = \{h_1 \dots h_{N_{AA}}\}, H^{AA} \subset H^A$ являются активными во время атаки. Динамика изменения активности зараженных хостов обуславливается следующими возможностями во время работы сети:

- Зараженный хост h_j ($h_j \in H^A$) может стать пользовательским $\left(H^A = H^A \backslash \{h_j\}, N_A = N_A - 1\right)$.
- Пользовательский хост h_i ($h_i \in H^U$) может стать зараженным $(H^A = H^A \cup \{h_i\}, H^U = H^U \backslash \{h_i\}, N_A = N_A + 1)$.
- Злоумышленник может приостанавливать и возобновлять атаку.
- Злоумышленник может менять количество *активных* хостов во время атаки.

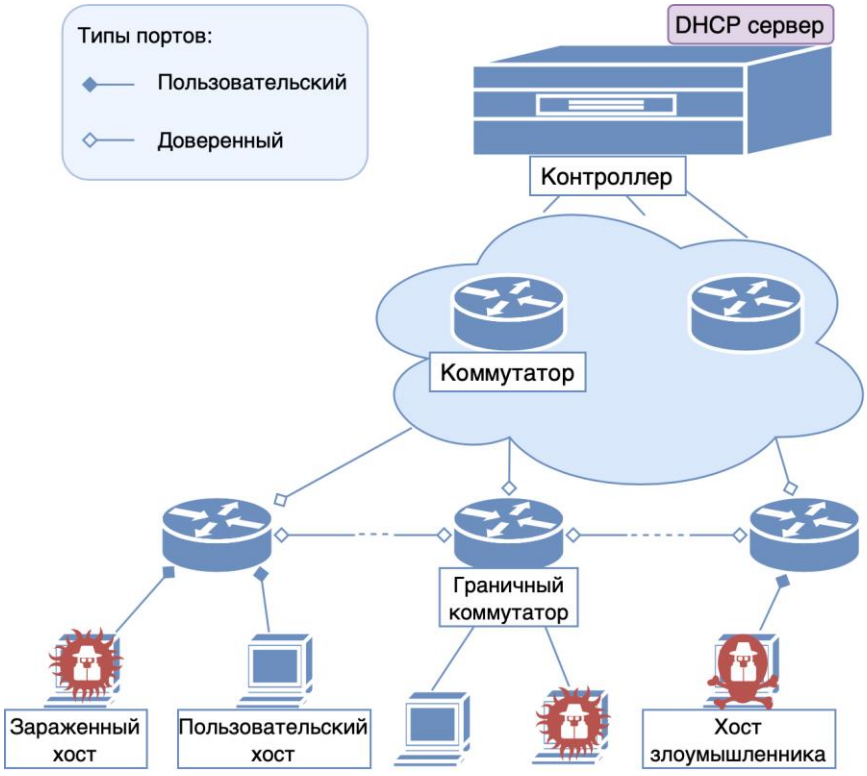

Рис. 4 Топология сети для решаемой задачи

Для каждого граничного коммутатора $S_j \in S_E$ будем разделять его множество портов $P_j = \{p_1, p_2, \dots, p_{N_j}\}$ следующим образом (Рис. 4):

- $P_j^T = \{p_{T_1}^T, \dots, p_{T_k}^T\}$ – доверенные порты (*trusted*), которые соединяют каналом связи j-й граничный коммутатор с сетевыми сервисами или другими коммутаторами;
- $P_j^H = \{p_{H_1}^H, \dots, p_{H_m}^H\}$ – хостовые порты (*hosts*);
- $P_j = P_j^T \cup P_j^H, \left(P_j^T \cap P_j^H\right) = \emptyset$.

Хостовые в свою очередь подразделяются на две группы:

- $\mathrm{P_j^I} = \{\mathrm{p_{I_1}}, \dots, \mathrm{p_{I_p}}\}$ – зараженные порты (*infected*);
- $\mathrm{P_j^U} = \{\mathrm{p_{U_1}}, \dots, \mathrm{p_{U_q}}\}$ – пользовательские порты (*users*);
- $P_j^H = \mathrm{P_j^I} \cup \mathrm{P_j^U}, (\mathrm{P_j^I} \cap \mathrm{P_j^U}) = \emptyset$.

Для каждого коммутатора $S_j \in S_E$ задается множество потоков в таблицах потоков коммутатора $\mathrm{F_j} = \left\{\mathrm{f_1}, \mathrm{f_2}, \dots, \mathrm{f_{n_j}}\right\}$. В зависимости от того, с какого типа порта поступают потоки, $\mathrm{F_j}$ подразделяется на множество пользовательских (поступающих с портов из $\mathrm{P_j^T}$ и $\mathrm{P_j^U}$) и множество фиктивных потоков (поступающих с портов из $\mathrm{P_j^I}$).

Необходимо:

- Определить множество зараженных хостов H^A.
- Определить множество правил R_1 для уменьшения фиктивных потоков.
- Определить множество правил R_2 для устранения последствий атаки (очистки таблиц потоков коммутаторов от правил для фиктивных потоков).

3 Обзор методов предотвращения DDoS атак

Существует различные методы по обнаружению и предотвращению DDoS атак на контроллер ПКС, которые могут быть классифицированы в зависимости от их ориентированности, согласно [7], на внутренние или внешние. Внутренне ориентированные методы подразумевают изменение логики функционирования контроллера и/или сетевых элементов ПКС. Внешне ориентированные методы основываются только на анализе потоков с хостов. Множество внешне ориентированных методов можно разделить на методы, основанные на машинном обучении, и статистические методы.

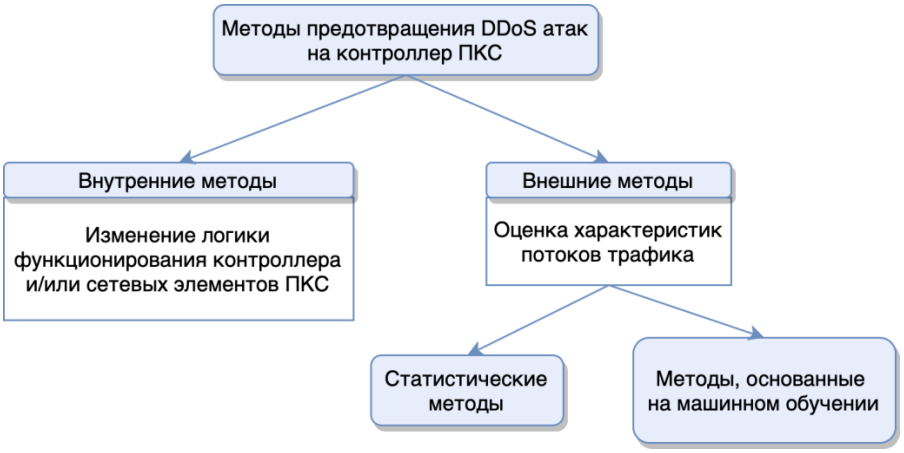

Рис. 5 Классификация методов предотвращения DDoS атак в ПКС

Критерии обзора методов:

1. Ориентированность метода (внутренний или внешний).
2. Топология (фиксированная или нефиксированная).
3. Назначение IP адресов (только статически или статически и по динамическому протоколу).
4. Динамика активности зараженных хостов (учитывается или нет).
5. Эффективность метода.
   5.1. Ошибка 1-го рода (ложное срабатывание).
   5.2. Ошибка 2-го рода (не распознание фиктивного трафика).
   5.3. Утилизация ЦПУ и ОП сервера контроллера (процентное соотношение утилизации без/с использования/ем метода).
   5.4. Задержка на установление нового потока.

3.1 Метод на основе разделения очереди обработки запросов в контроллере

Авторы метода [8] предлагают логически разделить очередь обработки запросов в контроллере на k очередей, каждая из которых соответствует одному коммутатору, имеющему канал связи с контроллером. Обслуживание этих очередей будет происходить в соответствии с какой-либо дисциплиной планирования, например, циклической. В итоге создается изолированное распределение вычислительной мощности контроллера для каждого коммутатора, что позволяет поддерживать функционирование контроллера во время DDoS атаки. Авторы называют предложенную модель «MultiQ».

Ориентированность метода. Разделение очереди обработки Packet-In является признаком внутренне ориентированного метода, требующего изменения очереди обработки запросов в контроллере.

Топология. Метод требует фиксированного количества коммутаторов, имеющих канал связи с контроллером. Другие параметры топологии (количество и местоположение хостов и остальных коммутаторов) может меняться.

Назначение IP адресов. Рассматриваемое решение совместимо с обоими методами назначения IP адресов.

Динамика активности зараженных хостов. Метод не различает динамики активности зараженных хостов, так как в нем отсутствует механизм обнаружения зараженных хостов.

Эффективность метода. В экспериментальном исследовании сравниваются три модели: «MultiQ», «SingleQ» (модель с одной очередью) и «Static» (модель с ограничением в коммутаторах на скорость отправки пакетов на контроллер). Задержка обработки и доля сброшенных пакетов измеряются на трех последовательно соединенных коммутаторов, первый из которых принимает на себя основной удар атаки.

Наименьшая доля сброшенных потоков для «MultiQ» на первом коммутаторе видна на слабых атаках (в 1,2 раза превышающие скорость обработки контроллера) – 20%; на более серьезных потеря примерно равна потере у «Static» – 50-90%. Но уже для второго коммутатора «MultiQ» показывает практически нулевую потерю, что гораздо лучше, чем у модели с одной очередью (30-90%). Таким образом, **ошибка первого рода** равна 0,2–0,9. **Ошибка второго рода** не была измерена экспериментально.

Метод позволяет серверу, на котором расположен контроллер, распределять утилизацию ресурсов между k логическими очередями. Пусть нагрузка на сервер во время нормальной работе сети равна u%. Тогда если DDoS атаке подвергается только один

коммутатор, имеющий связь с контроллером, то **утилизация ЦПУ и ОП** не будет выше, чем $\left(u - \frac{u}{k} + \frac{100}{k}\right)$ %. Если атаки подвержены все коммутаторы, то утилизация ЦПУ и ОП будет максимальной – 100%. Экспериментально утилизация ЦПУ и ОП не измерялась.

Задержка на установление нового потока в предложенной модели меньше, чем у «SingleQ», и примерно одинакова с «Static». На серьезных атаках, превышающих скорость обработки контроллера в 8 раз, «MultiQ» показывает лучший результат для первого коммутатора среди остальных – 18 секунд. Для остальных коммутаторов задержка меньше 5 секунд.

3.2 Метод на основе динамических функций поведения хостов в зависимости от количества запросов на установление новых потоков

Авторы метода SECO (SDN sEcure COntroller algorithm) [9] предлагают несколько функций для подсчета количества Packet-In от портов коммутаторов, к которым подключены хосты, и приходящих на порты контроллера за время t:

$$\gamma_{ij}(t) = \sum packet\_in_{ij}(t)\,, \beta_i(t) = \sum_{j=1}^{m} \gamma_{ij}(t), \tag{3-1}$$

где $i = \overline{1,n}$ – номер порта контроллера, $j = \overline{1,m}$ – номер порта коммутатора, к которому подключен хост.

Отталкиваясь от заданного максимального порогового значения утилизации ЦПУ контроллера σ%, экспериментально рассчитывается максимально допустимое общее количество Packet-In $\omega(t)$, которое ограничивает величины $\beta_i(t)$ для каждого порта контроллера:

$$\omega(t) = \sum_{i=1}^{n} \beta_i(t) \tag{3-2}$$

Изначально пороговые значения количества Packet-In для портов контроллера устанавливаются по умолчанию:

$$\alpha_i(t) = \frac{\omega(t)}{n} \tag{3-3}$$

Затем, если какой-либо порт превысил данный порог и общее количество Packet-In удовлетворяет допустимой утилизации ЦПУ:

$$\beta_i(t) > \alpha_i(t), \sum_{i=1}^{n} \beta_i(t) \leq \omega(t), \quad (3\text{-}4)$$

то пороговое значение $\alpha_i(t)$ обновляется: $\alpha_i(t) = \tau_i(T)$. Новые значения $\tau_i(T)$ рассчитываются каждые T секунд в зависимости от текущего количества Packet-In с i-го порта контроллера:

$$\tau_i(T) = \left\lfloor \omega(t) * \frac{\lambda_i(T)}{\sum_{i=1}^{n} \lambda_i(T)} \right\rfloor, \quad (3\text{-}5)$$

где $\lambda_i(T)$ – количество Packet-In, приходящих на порт контроллера за время T.

Если какой-либо порт превысил порог и общее количество Packet-In превысило порог для допустимой утилизации ЦПУ:

$$\beta_i(t) > \alpha_i(t), \sum_{i=1}^{n} \beta_i(t) > \omega(t), \quad (3\text{-}6)$$

то метод фиксирует начало DDoS атаки.

Авторы метода различают две возможные ситуации – когда заражен только хост и когда заражен коммутатор. После обнаружения начала атаки устанавливается правило сброса пакетов с таймаутом t_d и портом коммутатора, который превысил пороговое значение. Во время атаки продолжается расчет $\beta'_i(t')$ и $\alpha'_i(t')$ количества Packet-In за время t'. Когда $\beta'_i(t')$ начинает удовлетворять допустимой утилизации ЦПУ, то считается, что атака закончена. Если после установки правила сброса пакетов $\beta'_i(t')$ продолжает превышать порог, значит, заражен коммутатор, и устанавливается правило сброса пакетов от этого коммутатора.

Ориентированность метода. Метод направлен на оценку характеристик сетевых потоков, а именно количество Packet-In, приходящих от новых потоков, поэтому он является внешне ориентированным статистическим.

Топология. Метод подразумевает наличие фиксированной топологии.

Назначение IP адресов. Рассматриваемое решение совместимо с обоими методами назначения IP адресов.

Динамика активности зараженных хостов. Метод специализирован для настройки на динамическое поведение злоумышленника, так как допускает возможность перехода хоста как из категории пользовательских в категорию зараженных, так и наоборот.

Эффективность метода. В экспериментальном исследовании **ошибка первого рода** не была рассчитана. **Ошибка второго рода** (не распознание DDoS атаки) варьируется в зависимости от мощности DDoS атаки. Самая маленькая ошибка, равная

0.1–0.0, возникает при генерации пакетов с зараженных хостов с интервалом 60–100 мс. Ошибка второго рода возрастает до 0.7 при генерации пакетов с интервалом от 60 до 40 мс. При более сильной атаке защита метода не срабатывает.

Благодаря использованию метода **утилизация ЦПУ** контроллера во время атаки была снижена на 33%. **Утилизация ОП** контроллера в экспериментальном исследовании не измерялась.

Задержка на установление нового потока не была измерена в экспериментальном исследовании.

3.3 Метод на основе общей энтропии значений полей потоков

Метод, представленный в работе [10], имеет три фазы: номинальная, подготовительная, активная. На номинальной стадии в период работы сети без атак создаются номинальные профили трафика. В течение этого периода коммутатор отправляет заголовки всех пакетов на контроллер. Авторы выделяют следующие полезные поля в записи потока: IP адреса и порты отправителя и получателя ($IP_{src}, IP_{dst}, P_{src}, P_{dst}$), тип протокола, размер пакетов, TTL и TCP флаг (TCP_{flag}). Эти поля формируются парами A_i следующим образом (всего 28 пар): $A_1 = (IP_{src}, IP_{dst})$, $A_2 = (IP_{src}, P_{src}), \dots, A_8 = (IP_{dst}, P_{src}), \dots, A_{28} = (TTL, TCP_{flag})$.

Номинальный профиль (N_i) для пары полей A_i за период α ($\alpha = 1000$ количество пакетов, прошедших через коммутатор) – это матрица $m_i \times 3$ (Рис. 6), первые два столбца которой соответствуют содержанию полей в заголовке пакетов, а третий – количество пакетов, соответствующих этому содержанию полей (m_i – количество различных содержаний для пары полей A_i).

$$N^{\mathbb{A}_i} = \begin{bmatrix} x_1 \in \mathbb{A}_i^1 & y_1 \in \mathbb{A}_i^1 & PAC^{\mathbb{A}_i^1} \\ x_2 \in \mathbb{A}_i^2 & y_2 \in \mathbb{A}_i^2 & PAC^{\mathbb{A}_i^2} \\ .. & .. & .. \\ .. & .. & .. \\ x_{m_i} \in \mathbb{A}_i^{m_i} & y_{m_i} \in \mathbb{A}_i^{m_i} & PAC^{\mathbb{A}_i^{m_i}} \end{bmatrix}$$

Рис. 6 Матрица номинального профиля для пары полей A_i

Для каждого из 28 номинальных профилей рассчитывается общая энтропия JN_{N_i}:

$$JN_{N_i} = -\sum_{k=1}^{m_i} \frac{PAC_k^{A_i}}{\alpha} * \log\left(\frac{PAC^{A_i}}{\alpha}\right), \tag{3-7}$$

где $PAC_k^{A_i}$ – количество пакетов в k-той строке матрицы номинального профиля N_i, $\alpha = 1000$ количество пакетов в периоде.

Когда пропускная способность на коммутаторе превышает заранее заданное значение, активируется подготовительная фаза. На ней коммутатор продолжает отсылать в контроллер заголовки всех пакетов, для которых рассчитывается аналогично номинальной стадии *текущие профили* для пар полей и *общая энтропия* текущих профилей JC_{N_i}. Затем оба значения сравниваются и находится профиль с максимальным отклонением:

$$\Delta J_{N_i} = JC_{N_i} - JN_{N_i}, \tag{3-8}$$

$$\Delta J_{MAX} = max\{\Delta J_{N_1}, \dots, \Delta J_{N_{28}}\} \tag{3-9}$$

Пара полей, соответствующая номинальному профилю с максимальным отклонением энтропии ΔJ_{MAX}, помечается как «*подозрительная пара*».

После нахождения подозрительной пары метод переходит в активную фазу. На ней пересылка заголовков пакетов в контроллер может создать дополнительную нагрузку на контроллер, поэтому профиль для подозрительной пары рассчитывается сразу на коммутаторе и на контроллер отправляется только это значение в конце каждого периода α. Затем рассчитываются оценки β как частное текущего и нормального профилей для подозрительной пары каждой записи потока: $\beta = SC_i/SN_i$. Все оценки заносятся в таблицу оценок и далее используются для расчета функции распределения текущего порогового значения (θ_C) с помощью алгоритма сброса нагрузки:

$$CDF(\theta_C) = \Phi, \tag{3-10}$$

$$1 - \Phi = \frac{\phi}{\varphi}, \tag{3-11}$$

где ϕ – приемлемое количество пакетов, φ – текущий количество входящих пакетов, Ф – доля пакетов, которая должна быть сброшена. Пороговое значение для подозрительной пары рассчитывается, как $\theta_S = (\theta_C + \theta_P)/2$, где θ_P – пороговое значение за предыдущий период α. Если оценка для записи потока превышает порог $\beta > \theta_S$, то пакеты от потока будут сброшены, иначе – пакеты пересылаются получателю. На последнем шаге сформированные правила сброса и пересылки сравниваются с уже имеющимися на коммутаторе и отправляется только отличающиеся правила.

Ориентированность метода. Метод основан на расчете энтропии значений полей пакетов потоков и является внешне ориентированным статистическим.

Топология. Топология может меняться динамически.

Назначение IP адресов. Рассматриваемое решение совместимо с обоими методами назначения IP адресов.

Динамика активности зараженных хостов. Метод допускает настройку на динамическое поведение злоумышленника, так как все пороговые значения рассчитываются динамически.

Эффективность метода. Наибольшее значение **ошибки первого рода** 0,02 и **ошибки второго рода** 0,18.

Утилизация ЦПУ и **ОП** контроллера в экспериментальном исследовании не измерялась.

Задержка на установление нового потока не возрастает во время противодействия атаке, так как анализ поведения потока и решение о сбросе пакетов потока происходит после установки правила для пересылки.

3.4 Выводы

В проведенном обзоре было рассмотрено три метода предотвращения DDoS атак на контроллер ПКС (Таблица 1).

В методе на основе логического разделения очереди обработки запросов на контроллере [8] можно выделить существенный недостаток в том, что для реализации метода необходимо вносить изменения в логику функционирования контроллера. Данный метод не позволяет производить автоматическую перенастройку, если произойдет отключение какого-либо из коммутаторов, имеющих канал связи с контроллером. Метод также не учитывает динамическое поведение злоумышленника во время проведения атаки. Достоинством метода является то, что он позволяет производить смену IP адресов хостов в сети как статически, так и по DHCP протоколу.

Метод на основе общей энтропии [10] учитывает динамику и в топологии, и в поведении злоумышленника, однако нельзя в полной мере оценить эффективность метода, так как приведено только измерение его точности (ошибка 1-го рода равна 0,02; ошибка 2-го рода – 0,18), но экспериментально не проверена утилизация CPU и RAM контроллера.

Наиболее полезным для дальнейшей разработки собственного метода был выделен метод на основе динамических функций поведения хостов [9]. А именно применение формул (3-4), (3-5) для пересчета пороговых значений триггеров для обнаружения начала атаки в зависимости от изменяющегося количества трафика для конкретных хостов. Главным параметром для обнаружения атаки согласно формуле (3-6) является превышение порогового значения утилизации ЦПУ контроллера, что позволяет корректно реагировать не только на изменение активности зараженных хостов, но и на медленную DDoS атаку, подразумевающую большое число зараженных хостов при небольшой

скорости генерации Packet-In с каждого из них. Также в данном методе правила сброса пакетов от хостов, классифицированных как зараженные, устанавливается с таймаутом, что позволяет учитывать возможность перехода зараженного хоста в класс пользовательских, если злоумышленник теряет к нему доступ. Недостатком используемых в этом методе формул является то, что топология сети должна быть фиксированной.

	Ориент-ть метода	**Топология**	**Назначение IP адресов**	**Динамика активности зараженных хостов**
Метод на основе логического разделения очереди обработки запросов в контроллере [8]	Внутренний	Допустимо изменение количества и местоположения хостов и коммутаторов, за исключением коммутаторов, имеющих канал связи с контроллером	Статически и по динамическому протоколу	Не учитывается
Метод на основе динамических функций поведения хостов [9]	Внешний	Фиксированная	Статически и по динамическому протоколу	Учитывается
Метод на основе общей энтропии значения полей потоков [10]	Внешний	Допустимо динамическое изменение топологии	Статически и по динамическому протоколу	Учитывается

Таблица 1.1 Результаты обзора методов противодействия DDoS атакам

	Ошибка первого рода	**Ошибка второго рода**	**Утилизация CPU контроллера**	**Утилизация RAM контроллера**	**Задержка на установление нового потока**
Метод на основе логического разделения очереди обработки запросов в контроллере [8]	0,2 - 0,9	-	Для k логический очередей и u% утилизации CPU при отсутствии атаки утилизация во время атаки $< \left(u - \frac{u}{k} + \frac{100}{k}\right)\%$.	-	5 - 18 сек
Метод на основе динамических функций поведения хостов [9]	0,1 - 0,7	-	Снижена на 33%	-	-
Метод на основе общей энтропии значения полей потоков [10]	0,02	0,18	-	-	Совпадает с задержкой в отсутствии атаки

Таблица 1.2 Результаты обзора методов противодействия DDoS атакам

4 Разработка метода предотвращения DDoS атак на ПКС-контроллер

Для решения задачи предотвращения DDoS атаки на контроллер ПКС декомпозируем ее на следующие подзадачи:

1. Обнаружение атаки.
   1.1. Определить критерий начала атаки.
2. Приостановка атаки.
   2.1. Определить критерий классификации хостов.
   2.2. Определить действия для уменьшения фиктивных потоков с хостов, классифицированных как зараженные.
   2.3. Определить критерий конца атаки.
3. Устранение последствий атаки.
   3.1. Определить действия для очистки таблиц потоков коммутаторов от правил для фиктивных потоков.

Помимо решения представленных подзадач метод должен учитывать динамические изменения в топологии сети, а также изменения активности зараженных хостов.

4.1 Описание работы метода

Для учета возможности динамического изменения топологии сети метод производит **мониторинг состояния сети**. Для этого создается «*таблица соответствия хостов коммутаторам*» в сети, в которой MAC адреса хоста ($hMAC_i, i = \overline{1, N_H}$) является ключом, а элемент данных содержит соответствующий IP адрес ($hIP_j, j = \overline{1, N_H}$), идентификатор ($DPID_k, k = \overline{1, N_E}$) и номер физического порта коммутатора ($NP_{k,l}, k = \overline{1, N_E}, l = \overline{1, N_k}$), к которому хост имеет подключение, и статус хоста: подключен или отключен от сети (Таблица 2).

MAC адрес	IP адрес	DPID	Порт	Статус
$hMAC_1$	hIP_1	$DPID_{k_1}$	PN_{k_1,l_1}	ON
…	…	…	…	…
$hMAC_{N_H}$	IP_{N_H}	$DPID_{k_{N_H}}$	$PN_{k_{N_H},l_{N_H}}$	ON

Таблица 2 Пример заполненной «таблицы соответствия хостов коммутаторам» в сети

После занесения хоста в «таблицу соответствия» в коммутатор, чей номер указан в данной таблице, устанавливаются правила фильтрации (Таблица 3). В результате на граничных коммутаторах сбрасываются потоки, если MAC адрес их источника отсутствует или не совпадает с IP адресом и/или номером входного порта, хранящихся в «таблице соответствия».

DPID	Match	Instructions	Command	Table_ id	Idle_ timeout	Hard_ timeout
$\boldsymbol{DPID_{k_1}}$	in_port = $\boldsymbol{PN_{k_1,l_1}}$, priority = 3, eth_type = 0x800, eth_src = $\boldsymbol{hMAC_1}$, ipv4_src = $\boldsymbol{hIP_1}$	GoToTable (1)	OFPFC_ ADD	0	0	0
	…	GoToTable (1)	OFPFC_ ADD	0	0	0
$\boldsymbol{DPID_{k_{N_H}}}$	in_port = $\boldsymbol{PN_{k_{N_H},l_{N_H}}}$, priority = 3, eth_type = 0x800, eth_src = $\boldsymbol{hMAC_{N_H}}$, ipv4_src = $\boldsymbol{IP_{N_H}}$	GoToTable (1)	OFPFC_ ADD	0	0	0
$\boldsymbol{DPID_{k_1}}$	in_port = $\boldsymbol{PN_{k_1,l_1}}$, priority = 3, eth_type = 0x806, eth_src = $\boldsymbol{hMAC_1}$	GoToTable (1)	OFPFC_ ADD	0	0	0
	…	GoToTable (1)	OFPFC_ ADD	0	0	0
$\boldsymbol{DPID_{k_{N_H}}}$	in_port = $\boldsymbol{PN_{k_{N_H},l_{N_H}}}$, priority = 3, eth_type = 0x806, eth_src = $\boldsymbol{hMAC_{N_H}}$	GoToTable (1)	OFPFC_ ADD	0	0	0
$\boldsymbol{DPID_{k_1}}$	in_port $\boldsymbol{PN_{k_1,l_1}}$, priority = 1, eth_type = 0x800	DropAction	OFPFC_ ADD	0	0	0
	…	DropAction	OFPFC_ ADD	0	0	0
$\boldsymbol{DPID_{k_{N_H}}}$	in_port = $\boldsymbol{PN_{k_{N_H},l_{N_H}}}$, priority = 2, eth_type = 0x800	DropAction	OFPFC_ ADD	0	0	0
$\boldsymbol{DPID_{k_1}}$	in_port = $\boldsymbol{PN_{k_1,l_1}}$, priority = 1, eth_type = 0x806	DropAction	OFPFC_ ADD	0	0	0
	…	DropAction	OFPFC_ ADD	0	0	0
$\boldsymbol{DPID_{k_{N_H}}}$	in_port $\boldsymbol{PN_{k_{N_H},l_{N_H}}}$, priority = 2, eth_type = 0x806	DropAction	OFPFC_ ADD	0	0	0

Таблица 3 Правила фильтрации по «таблице соответствия хостов коммутаторам»

При появлении нового хоста в сети информация о нем заносится в новое поле «таблицы соответствия» и устанавливаются правила фильтрации на коммутаторы. Если хост отключился от сети, то в колонке «статус» поле меняется на «OFF». При обнаружении смены IP адреса через DHCP-сервер или статически вносятся изменения в данные о хосте в «таблице соответствия», удаляются старые правила фильтрации для этого хоста и устанавливаются новые. При обнаружении миграции хоста на другой коммутатор и/или другой порт вносятся изменения в поля соответствующей записи в «таблице соответствия».

4.1.1 Решение подзадачи обнаружения атаки

Введем необходимые величины для определения критерия начала атаки. Из формальной постановки задачи известны:

- $S_E = \{S_1 \dots S_{N_E}\}$ – множество граничных коммутаторов.
- $P_j^H = \mathrm{P_j^I} \cup \mathrm{P_j^U}, (\mathrm{P_j^I} \cap \mathrm{P_j^U}) = \emptyset$ – множество хостовых портов на j-ом коммутаторе, разделенных на множества зараженных и множество пользовательских.
- $H = H^A \cup H^U, (H^A \cap H^U) = \emptyset$ – множество хостов в сети, где H^A – множество зараженных хостов, H^U – множество пользовательских хостов.

Заранее экспериментально рассчитывается $\omega(t)$ – количество Packet-In в сети за время t, которое может привести к перегрузке контроллера (то есть при получении такого количества Packet-In за время t утилизация ЦПУ контроллера достигнет σ%, где σ устанавливается согласно необходимым требованиям к утилизации CPU). Для каждого хоста из множества H^U подсчитывается количество Packet-In за время t (**4-1)**, а для хостов из множества H^A – количество пакетов, поступивших с этого хоста за время t (4-1.1).

$$\gamma_{ij}(t) = \sum packet\_in_{ij}(t), i \in \{1, .., N_E\}, j \in P_i^U \quad (4\text{-}1)$$

$$\gamma'_{ij}(t) = \sum packetNum_{ij}(t), i \in \{1, .., N_E\}, j \in P_i^I \quad (4\text{-}1.1)$$

Для каждого i-го коммутатора подсчитывается и хранится следующая информация:

- Количество Packet-In за время t:

$$\beta_i(t) = \sum_{j \in P_i^U} \gamma_{ij}(t) + \sum_{j \in P_i^I} \gamma'_{ij}(t) \quad (4\text{-}2)$$

- Пороговое значение для количества Packet-In за время t:

$$\alpha_i(t) = \frac{\omega(t)}{n} \tag{4-3}$$

- Количество Packet-In за время T:

$$\lambda_i(T) = \sum_{j=1}^{N_i} \gamma_{ij}(T) \tag{4-4}$$

- Новое пороговое значение за время T (где $T = 3t$):

$$\tau_i(T) = \left\lfloor \omega(t) \frac{\lambda_i(T)}{\sum \lambda_i(T)} \right\rfloor \tag{4-5}$$

При изменении количества работающих граничных коммутаторов (N_E) пороговые значения $\alpha_i(t)$ пересчитываются.

Через заданный промежуток времени t проверяются два условия:

1) Превышает ли количество Packet-In за время t с i-го коммутатора свое пороговое значение:

$$\beta_i(t) >_? \alpha_i(t) \tag{4-6}$$

2) Превышает ли общее количество Packet-In со всех коммутаторов свое пороговое значение:

$$\sum_{i=1}^{N_E} \beta_i(t) >_? \omega(t) \tag{4-7}$$

Если для i-го коммутатора условия (4-6) и (4-7) не выполнены, то проверяется множество зараженных портов данного коммутатора P_i^I. Если оно не пусто, то все хосты из множества зараженных хостов, подключенные к i-му коммутатору, перемещаются в множество пользовательских хостов и $P_i^U = P_i^U \cup P_i^I$, $P_i^I = \emptyset$.

Если выполнено (4-6), но не выполнено (4-7) то пороговые значения для всех коммутаторов обновляются:

$$\alpha_i(t) = \tau_i(T), i = \overline{1, N_E} \tag{4-8}$$

Если выполнено (4-6) и (4-7), то фиксируется начало атаки. Таким образом, **критерий начала атаки**:

$$\left(\beta_i(t) > \alpha_i(t)\right) \cap \left(\sum_{i=1}^{N_E} \beta_i(t) > \omega(t) \right), \forall i \in \{1, \dots, N_E\} \tag{4-9}$$

4.1.2 Решение подзадачи приостановки атаки

После обнаружения начала атаки через заданный промежуток времени (τ) запрашивается статистика с коммутаторов, после чего рассчитывается среднее количество пакетов на поток для каждого хоста (Packet Number per Flow):

$$PNF(\tau_i) = \frac{\sum_{flow} PacketNumber(\tau_i)}{FlowNumber(\tau_i)} \quad (4\text{-}10)$$

Для расчета оценки поведения хоста берется частное среднего количества пакетов на поток на количество Packet-In за время τ (где $\tau = \frac{t}{k}$):

$$score_i = \frac{PNF(\tau_i)}{\sum packet\_in(\tau_i)}, \sum packet\_in(\tau_0) = \frac{\gamma(t)}{k} \quad (4\text{-}11)$$

Оценка корректируется с помощью фактора забывания α:

$$score_i = score_i * (1 - \alpha) + score_{i-1} * \alpha \quad (4\text{-}12)$$

Для определения критерия классификации разделим множество хостов H на три подмножества $H = USR \cup INF \cup AMB, (USR \cap INF) \cup (INF \cap AMB) \cup (AMB \cap USR) = \emptyset$, где *USR* – множество пользовательских, *INF* – множество зараженных и *AMB* – множество неопределенных хостов. Множество *AMB* введено с целью уменьшения ошибок ложных срабатываний. При попадании хоста в данное множество делается дополнительная проверка: текущее значение утилизации CPU контроллера сравнивается с крайним наивысшим значением допустимой утилизации *TCU* ($\sigma < TCU < 100$). Если текущее превосходит *TCU*, то хост перемещается в множество зараженных, иначе остается во множестве неопределенных. Это означает, что потоки с него продолжают обрабатываться аналогично пользовательским потокам, так как контроллер пока способен их обработать. При дальнейших итерациях хосты из множества *AMB* будут перемещены в одно из множеств *USR, INF*.

Критерием классификации хостов является сравнение значений оценок поведения с верхним (TH) и нижним (TL) пороговыми значениями (подробнее о расчете пороговых значений описано в пункте 4.2 данной работы). в результате которого хосты классифицируются как *пользовательские*, *зараженные* или *неопределенные* в зависимости от попадания в соответствующий интервал распределения значений оценки поведения хостов (Рис. 7).

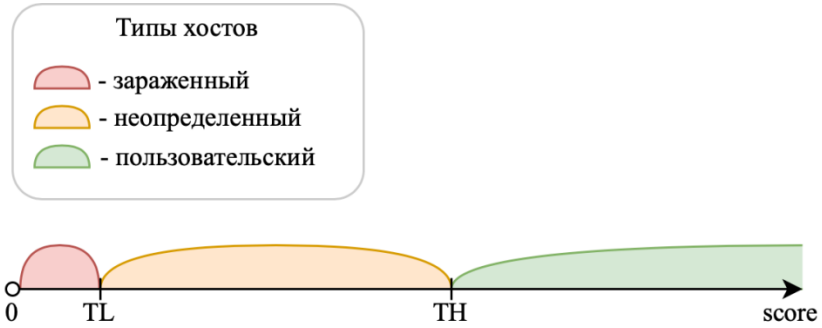

Рис. 7 Критерий классификации хостов

Пусть хост с номером n попал в множество зараженных. Тогда для него устанавливаются правила второго этапа фильтрации (Таблица 4). Чтобы учесть возможность перехода зараженного хоста во множество пользовательских, в правилах устанавливается мягкий таймаут, который автоматически удалит правило сброса, если с зараженного хоста не будет поступать пакеты в течении минуты.

DPID	Match	Instructions	Command	Table_ id	Idle_ timeout	Hard_ timeout
$\boldsymbol{DPID_{k_n}}$	in_port = $\boldsymbol{PN_{k_n,l_n}}$, priority = 4, eth_type = 0x800, eth_src = $\boldsymbol{hMAC_n}$, ipv4_src = $\boldsymbol{hIP_n}$	DropActio n	OFPFC_ ADD	0	60	0
$\boldsymbol{DPID_{k_n}}$	in_port = $\boldsymbol{PN_{k_n,l_n}}$, priority = 4, eth_type = 0x806, eth_src = $\boldsymbol{hMAC_n}$	DropActio n	OFPFC_ ADD	0	60	0

Таблица 4 Правила для фильтрации по критерию классификации

Когда общее количество Packet-In со всех коммутаторов перестает превышать свое пороговое значение, а все хосты классифицированы на пользовательские и зараженные, считается, что предотвращение атаки завершено. Таким образом, **критерий конца атаки**:

$$(AMB = \emptyset) \cap \left(\sum_{i=1}^{N_E} \beta_i(t) \leq \omega(t) \right) \quad (4\text{-}13)$$

4.1.3 Решение подзадачи устранения последствий атаки

Для очистки таблиц записей коммутаторов от записей фиктивных потоков во все коммутаторы из множества $S = \{S_1 \dots S_{N_S}\}$ рассылаются правила, которые удаляют записи

потоков, в которых MAC адреса отправителя или получателя совпадают с MAC адресами хостов из множества зараженных $H^A = \{h_{a_1}, \dots, h_{a_{NI}}\}$ (Таблица 5).

DPID	Match	Command	Table_id	Out_port	Out_group
$\boldsymbol{DPID_{k_{a_1}}}$	priority = 2, eth_type = 0x800, eth_src = $\boldsymbol{hMAC_{a_1}}$	OFPFC_ DELETE	OFPTT_ALL	OFPP _ANY	OFPP _ANY
	…	OFPFC_ DELETE	OFPTT_ALL	OFPP _ANY	OFPP _ANY
$\boldsymbol{DPID_{k_{a_{NI}}}}$	priority = 2, eth_type = 0x800, eth_src = $\boldsymbol{hMAC_{N_H}}$	OFPFC_ DELETE	OFPTT_ALL	OFPP _ANY	OFPP _ANY
$\boldsymbol{DPID_{k_{a_1}}}$	priority = 2, eth_type = 0x806, eth_src = $\boldsymbol{hMAC_{a_1}}$	OFPFC_ DELETE	OFPTT_ALL	OFPP _ANY	OFPP _ANY
	…	OFPFC_ DELETE	OFPTT_ALL	OFPP _ANY	OFPP _ANY
$\boldsymbol{DPID_{k_{a_{NI}}}}$	priority = 2, eth_type = 0x806, eth_src = $\boldsymbol{hMAC_{N_H}}$	OFPFC_ DELETE	OFPTT_ALL	OFPP _ANY	OFPP _ANY

Таблица 5 Правила для очистки таблиц коммутаторов от правил фиктивных потоков

4.2 Теоретический расчет пороговых значений для параметров метода

Для обеспечения корректной работы метода необходимо произвести анализ изменения значений оценки поведения хостов. Пусть сеть выдерживает установку минимум одного соединения с каждого хоста в 1 секунду. Одно успешно установленное соединение подразумевает добавление двух правил потока в таблицы коммутаторов. Для установки соединения в контроллере RUNOS 2.0 хост, инициирующий соединение, отправляет 7 Packet-In; хост на другой стороне соединения отправляет 4 Packet-In. Таким образом, в ситуации, когда каждый хост устанавливает по 1 соединению в 1 секунду, с каждого хоста отправляется 11 Packet-In. Тогда минимальное суммарное число Packet-In за время $t = k\tau$, приводящее к перегрузке контроллера, должно быть:

$$t * N_H * 11 = k * \tau * N_H * 11 < \omega, \qquad \min \omega = k * \tau * N_H * 11 + 1 \tag{4-14}$$

где N_H – число хостов в сети.

Найдем M_a – минимальное число Packet-In с каждого зараженного хоста, приводящее к перегрузке контроллера. Рассмотрим наихудший возможный вариант, когда все хосты в сети являются зараженными, тогда:

$$M_a \geq \frac{\omega}{N_H * k} \geq \frac{k * \tau * N_H * 11}{N_H * k} = 11\tau\,, \qquad \min M_a = 11\tau \tag{4-15}$$

Найдем верхнее пороговое значение оценки поведения зараженных хостов. Среднее количество пакетов на поток возьмем равным пяти ($PNF = 5$), как среднее для пользовательских потоков согласно [11]. Также предположим, что не было установлено ни одного фиктивного потока (у которых $PNF < 5$), в противном случае оценка была бы меньше максимальной TH.

$$TH \leq \frac{PNF}{11\tau} \leq \frac{5}{11\tau}, \qquad \max TH = \frac{5}{11\tau} \tag{4-16}$$

Найдем M_U – максимальное число Packet-In с пользовательского хоста. Рассмотрим случай, когда все хосты в сети пользовательские, тогда согласно (4-14):

$$M_U \leq \frac{\omega}{N_H * k} \leq \frac{k * \tau * N_H * 11}{N_H * k} = 11\tau\,, \qquad \max M_U = 11\tau \tag{4-17}$$

Теперь найдем нижнее пороговое значение оценки поведения пользовательских хостов. Возьмем среднее количество пакетов на поток равным единице ($PNF = 1$), как наименьшее из всех возможных *PNF*, тогда:

$$TL \geq \frac{PNF}{11\tau} \geq \frac{1}{11\tau}\,, \min TL = \frac{1}{11\tau} \tag{4-18}$$

Таким образом, распределение значений оценок хостов для корректной работы критерия классификации представлено на рисунке (Рис. 8).

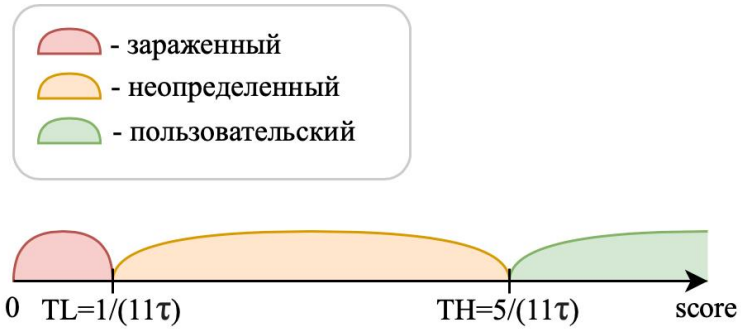

Рис. 8 Критерий классификации хостов

5 Разработка приложения для ПКС-контроллера RUNOS 2.0

5.1 Контроллер RUNOS 2.0

Программная реализация метода предотвращения DDoS атак на контроллер в ПКС/OpenFlow сети на основе оценки поведения хостов выполнена в виде приложения *DDoS_Defender* для контроллера RUNOS 2.0 [13].

ПКС-контроллер RUNOS 2.0 реализован на языке C++ с использованием механизмов многопоточности (Qt 5 и Boost.ASIO). Он основан на концепции микросервисов, реализованных в виде приложений, отвечающих за свою функцию. Приложение может генерировать *Qt сигналы*, которые могут быть распознаны другими приложениями. Для этого нужно получить объект приложения, от которого необходимо поймать сигнал, используя загрузчик приложений *Loader*. С помощью объекта приложения можно также получать доступ к его методам.

Все приложения, работающие на контроллере, разделены на приложения-сервисы ядра контроллера и пользовательские приложения. Сервисы ядра (такие как *Controller*, *OFMsgSender*, *LinkDiscovery*, *SwitchManager* и др.) обеспечивают работоспособность контроллера и выполняют базовые функции управления и контроля состояния сети. *Controller* предоставляет функциональность для регистрации функций-обработчиков пакетов, приходящих на контроллер. Благодаря этому любое приложение, развернутое на контроллере, может просматривать и анализировать пакеты из входного буфера контроллера. Приложение *OFMsgSender* выполняет низкоуровневые операции для установки и поддержки соединения с коммутаторами и обмена OpenFlow сообщениями и предоставляет возможность напрямую отправлять сообщения коммутаторам. *LinkDiscovery* отслеживает состояние и хранит информацию о каналах связи сети. *SwitchManager* контролирует состояния коммутаторов и уведомляет об их изменениях с помощью Qt сигналов.

Пользовательские приложения могут быть установлены посредством загрузки исходного кода приложения и пересборки контроллера. Параметры каждого пользовательского приложения должны быть предоставлены в json файле, которые автоматически учитываются при пересборке контроллера.

В данной версии контроллера RUNOS 2.0 появилась возможность использования базы данных Redis, работающей с данными в формате «ключ-значение». Приложение ядра *DatabaseConnector* предоставляет интерфейс работы с базой данных.

5.2 Описание архитектуры приложения DDoS_Defender

Для реализации DDoS_Defender понадобились объекты приложений ядра *Controller*, *OFMsgSender*, *LinkDiscovery*, *SwitchManager*. От приложения *LinkDiscovery* перехватываются сигналы об изменениях состояния каналов связи (LinkDiscovered), от *SwitchManager* – сигналы об изменениях состояния коммутаторов (SwitchUP, SwitchDown, LinkUp, LinkDown). С помощью приложения *Controller* регистрируются 3 функции обработчика для трех типов пакетов: Packet-In, Flow-Removed и MultipartReplyFlow для сбора статистики с коммутаторов.

Среди пользовательских приложений для реализации DDoS_Defender потребовались приложения *HostManager* и *DhcpServer*. От приложения *HostManager* перехватывается сигнал о появлении нового хоста в сети, их MAC и IP адреса; от *DhcpServer* – сигнал о смене IP адреса хоста по DHCP протоколу, MAC адрес хоста и новый IP адрес.

Входными данными для приложения являются интервал времени для сбора статистики во время атаки τ (tau), константа корректировки значения оценки поведения хостов α (alpha) и крайнее наивысшее пороговое значение для утилизации CPU контроллера TCU, использующееся во время классификации хостов.

Приложение состоит из четырех модулей:

- «Модуль мониторинга топологии» хранит «таблицу соответствия хостов коммутаторам» и отслеживает динамические изменения в топологии сети.
- «Модуль сбора информации о хостах» решает подзадачу обнаружения атаки и хранит таблицу, содержащую актуальное состояние собранной статистики о хостах.
- «Модуль классификации хостов» решает подзадачу приостановки атаки.
- «Модуль освобождения таблиц коммутаторов» решает подзадачу устранения последствий атаки.

Рассмотрим модель работы приложения (Рис. 9). После запуска сети начинают работать «модуль мониторинга топологии» и «модуль сбора информации о хостах». «Модуль мониторинга топологии» обрабатывает поступающие Packet-In (п.0),

перехватывает сигналы от приложений контроллера, вносит соответствующие изменения в «таблицу соответствия» (п.1), устанавливает корректные правила фильтрации (Таблица 3) с помощью обращения к приложению *OFMsgSender* (п.2) и сохраняет текущее состояние в базу данных (п.3). «Модуль сбора информации о хостах» подсчитывает приходящие пакеты Packet-In, каждые $t = 3\tau$ секунд отправляет сообщения MultipartRequestFlow коммутаторам для запроса статистики (п.4) и выполняет проверку условий (4-6) и (4-7). По результатам проверки может выполняться пересчет пороговых значений согласно (**4-8**) (п.5) , переход зараженного хоста во множество пользовательских (п.6) или может быть обнаружена атака (п.8). Каждые $T = 3t = 9\tau$ данные из таблицы поведения хостов в сети сохраняются в базу данных (п.7).

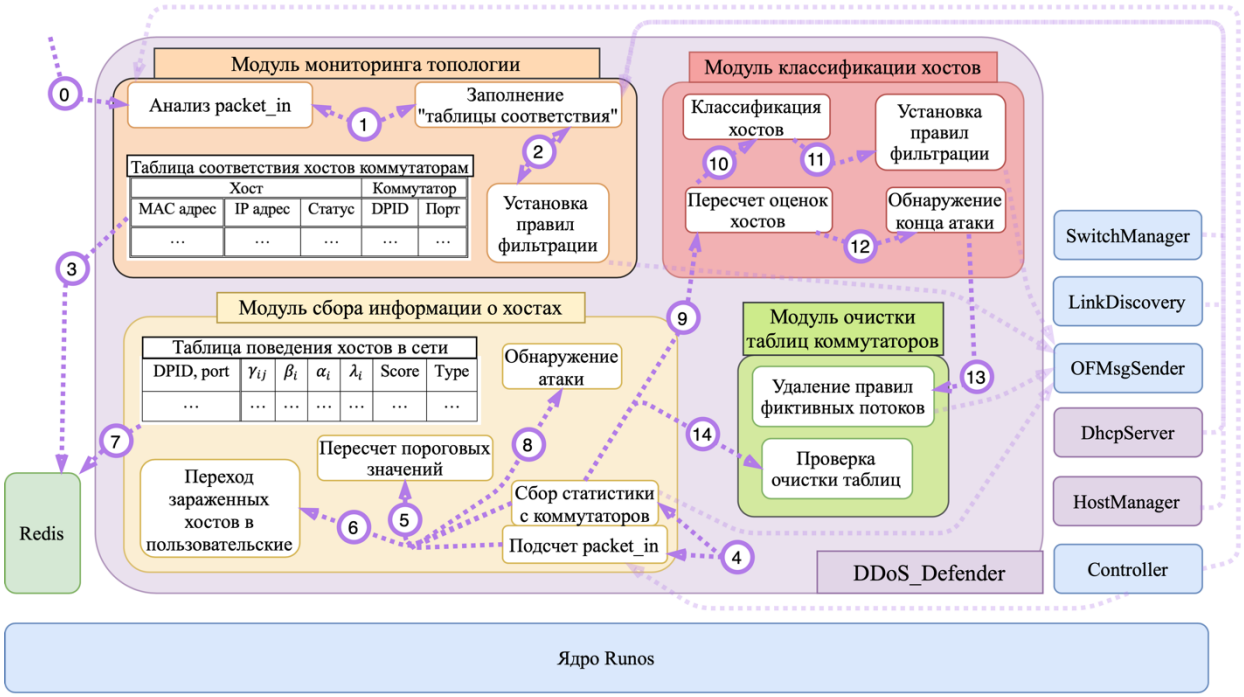

Рис. 9 Архитектура приложения DDoS_Defender

После обнаружения DDoS атаки в сети по критерию начала атаки (4-9) «модуль сбора информации о хостах» начинает собирать статистику с коммутаторов чаще – раз в τ секунд. Включается «модуль классификации хостов», который через каждые τ секунд пересчитывает оценки поведения хостов (п.9). Далее выполняется классификация хостов (п.10) и устанавливаются необходимые правила фильтрации (п.11).

Когда по критерию (4-13) обнаружен конец атаки (п.12), «модуль классификации хостов» выключается, «модуль сбора информации о хостах» начинает снова собирать статистику каждые $t = 3\tau$ секунд и включается «модуль освобождения таблиц коммутаторов». Последний рассылает правила (Таблица 5) во все коммутаторы (п.13) и проверяет в последующих сообщениях со статистикой с коммутаторов, остались ли

записи фиктивных потоков в таблицах (п.14). Когда таблицы перестанут содержать правил фиктивных потоков, «модуль освобождения таблиц коммутаторов» выключается.

5.3 Описание реализации приложения DDoS_Defender

Язык программирования: C++

Библиотеки: Redis

Файлы: DDoS_Defender.cc, DDoS_Defender.hpp, settings.json

Репозиторий проекта: https://github.com/anyaantipina/DDoS_Defender_1.0

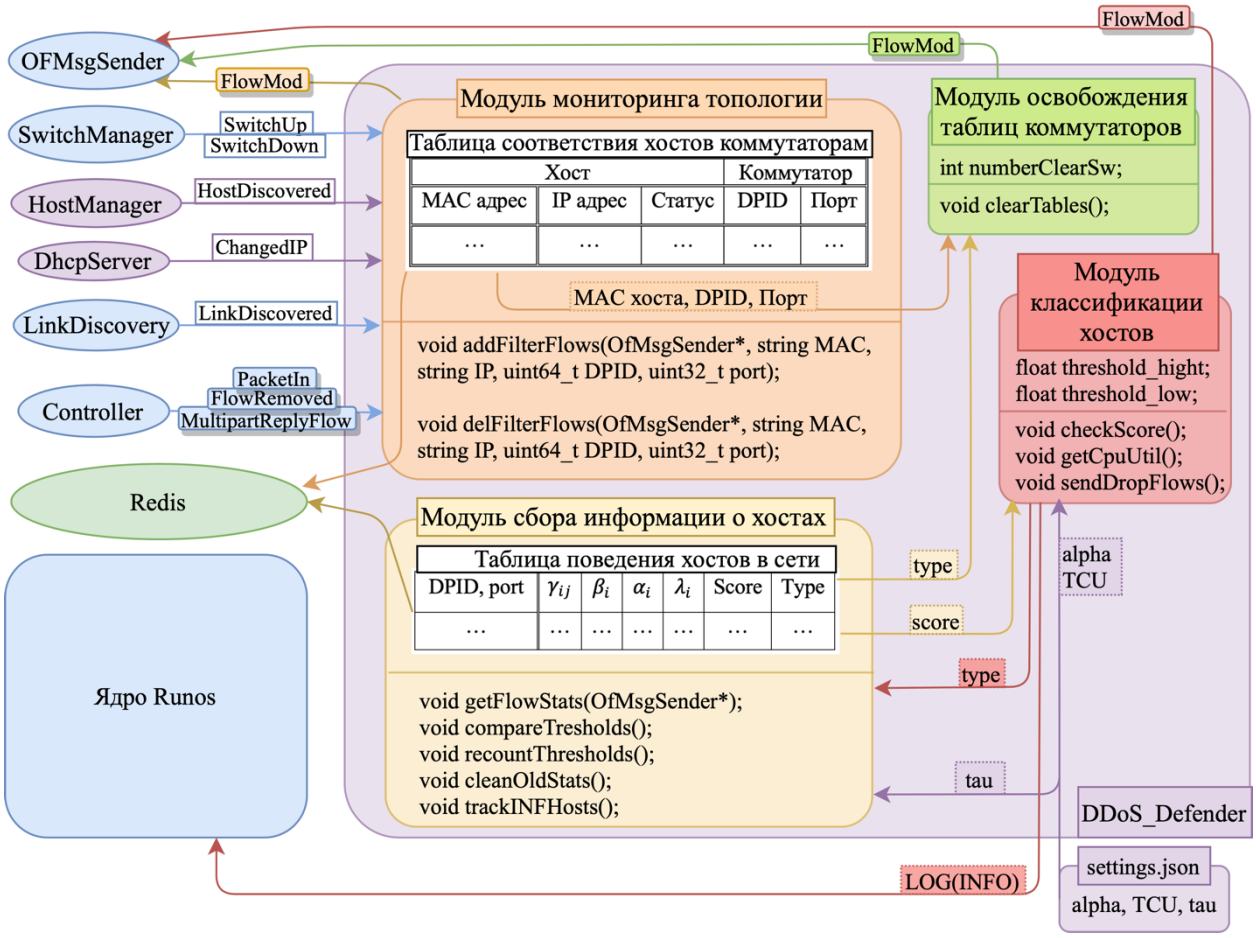

Рис. 10 Архитектура реализации приложения DDoS_Defender

Основные структуры реализации приложения DDoS_Defender представлены на рисунке (Рис. 10). Таблица соответствия хостов коммутаторам (*BindingTable*) реализована в виде хэш-таблицы std::unordered_map, где в качестве ключа используется MAC адрес хоста в строковом формате, а в элементе таблицы хранится структура со следующими полями:

- *MAC, IP* – адреса хоста в формате адресов (ethaddr, ipv4addr).
- *Status* – статус хоста в виде булевой переменной.
- *DPID, port* – идентификатор и номера порта коммутатора (uint64_t, uint32_t).

Таблица поведения хостов в сети также реализована в виде хэш-таблицы, где ключом является DPID коммутатора, а элемент-структура содержит следующие поля:

- *Beta, lambda* – количество Packet-In за t и T=3t секунд с коммутатора (4-2), (4-4).
- *Alpha* – пороговое значение для beta (4-3), (4-8).
- *Ports* – хэш-таблица хостовых портов: ключ – номер порта, элемент – структура:
  - *Gamma* – количество Packet-In за t секунд с хоста (**4-1**), (4-1.1).
  - *PNF* – среднее количество пакетов на поток с данного порта (4-10).
  - *Score* – оценка поведения хоста, подключенного к данному порту (4-11), (4-12).
  - *Type* – множество, к которому принадлежит хост, подключенный к данному порту (USR, INF, AMB).

5.4 Настраиваемые параметры приложения

Для установки и запуска приложения необходимо перейти в директорию /runos/src/apps, склонировать туда репозиторий приложения DDoS_Defender [18] и пересобрать контроллер.

Настраиваемые параметры задаются в файле settings.json в директории /runos/src/apps/ddos-defender:

- Alpha – константа корректировки оценки поведения хостов (4-12). Значение задается строкой, например, «0.5».
- TCU – верхнее пороговое значение утилизации CPU контроллера во время атаки. Значение задается строкой, например, «80.5».
- Tau – интервал сбора статистики во время атаки. Задается целочисленной константой.

После изменения параметров в файле settings.json необходимо выполнить команду «cmake ..» в директории /runos/build.

5.5 Формат выходных данных

Выходные данные работы приложения DDoS_Defender представлены в двух форматах. В лог контроллера выводятся сообщения о начале атаки, конце атаки и конце устранения последствий атаки, а также при переходе хоста из одного множества в другое ($H = USR \cup INF \cup AMB$, где H – множество всех хостов, USR – пользовательских хостов,

INF – зараженных хостов, AMB – неопределенных хостов) выводится сообщение, содержащее MAC и IP адреса данного хоста (Рис. 11).

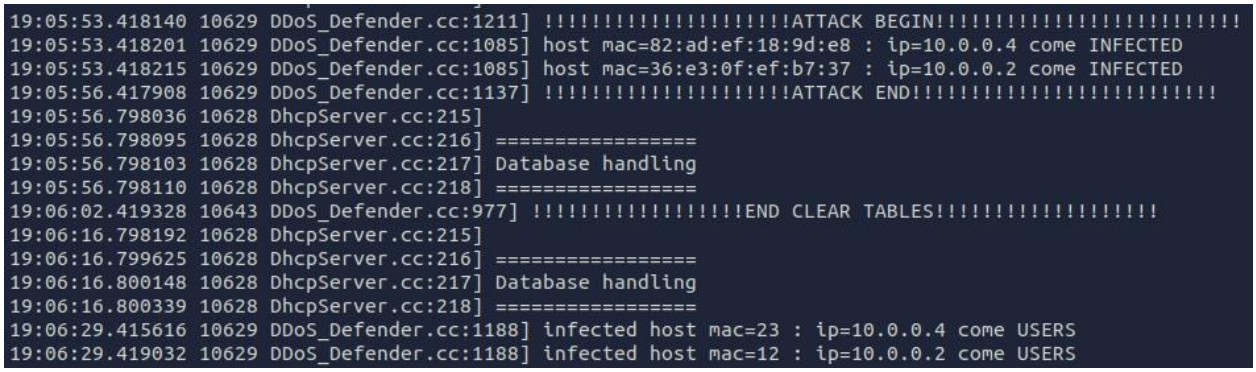

Рис. 11 Формат выходных данных работы приложения DDoS_Defender в лог RUNOS 2.0

В базу данных Redis периодически сохраняется состояние двух таблиц (Рис. 10). «Таблица соответствия хостов коммутаторам» доступна по ключу «ddos-defender:binding-table», через который можно получить ключи для каждой записи (порядковые номера) со всеми соответствующими полями, описанными в разделе 5.3. Таблица поведения хостов сохраняется аналогичным образом по ключу «ddos-defender:stats».

6 Экспериментальное исследование

Экспериментальное исследование направлено на оценку качества работы разработанного метода предотвращения DDoS атаки на контроллер, реализованного в виде приложения DDoS-Defender для контроллера RUNOS.

6.1 Методика

Тестирование метода проводится на топологиях с различным количеством хостов и долей зараженных хостов, способных производить миграцию между граничными коммутаторами. Стабильность работы метода исследуется на трех различных профилях активности зараженных хостов. Эффективность работы метода проверяется на шести динамически изменяющихся топологиях с различными характеристиками.

Исследование также включает в себя оценку скорости обнаружения, противодействия и устранения последствий атаки методом в зависимости от интервала сбора статистики с целью выявления его оптимального значения.

Перед проведением экспериментов для корректной работы метода рассчитывается количество Packet-In (ω), приводящее к утилизации ЦПУ контроллера более σ%. Для экспериментального исследования было выбрано значение утилизация $\sigma = 50\%$, которое достигается при $\omega = \cdots$.

6.1.1 Экспериментальный стенд

Для проведения экспериментального исследования используются:

- Эмулятор программно-конфигурируемой сети *Mininet 2.3.0d6* [12].
- OpenFlow-контроллер *RUNOS v2.0* [13].
- Пакетный генератор *Hyenae 0.36-1* [14].
- Утилита *iPerf3* [15].
- Утилита *psrecord*, которая использует библиотеку psutil для записи утилизации ЦПУ и памяти процесса [16].

В работе экспериментального стенда (Рис. 12) участвует одна хостовая машина, на которой запущены две виртуальные машины *Oracle VM VirtualBOX 6.0* [17] с характеристика, представленными в таблице (Таблица 6). На машине 2 запущен контроллер RUNOS на порте 6653, на машине 1 – скрипт Mininet с динамически изменяющейся топологией. Для автоматизации проведения экспериментов (разделы 6.1.2,

6.1.3, 6.1.4) на обеих машинах запускаются соответствующие скрипты, находящиеся в директории «test» [18]:

- dynamic_script.py, dynamic_script_RTT.py, dynamic_script_iPerf.py – тестирование качества работы метода;
- diff_topologies.sh, run_diff_top.sh – тестирование эффективности метода на топологиях с различными характеристиками;
- diff_attack_profiles.sh, run_diff_prof.sh – тестирование стабильности метода на разных профилях активности зараженных хостов.

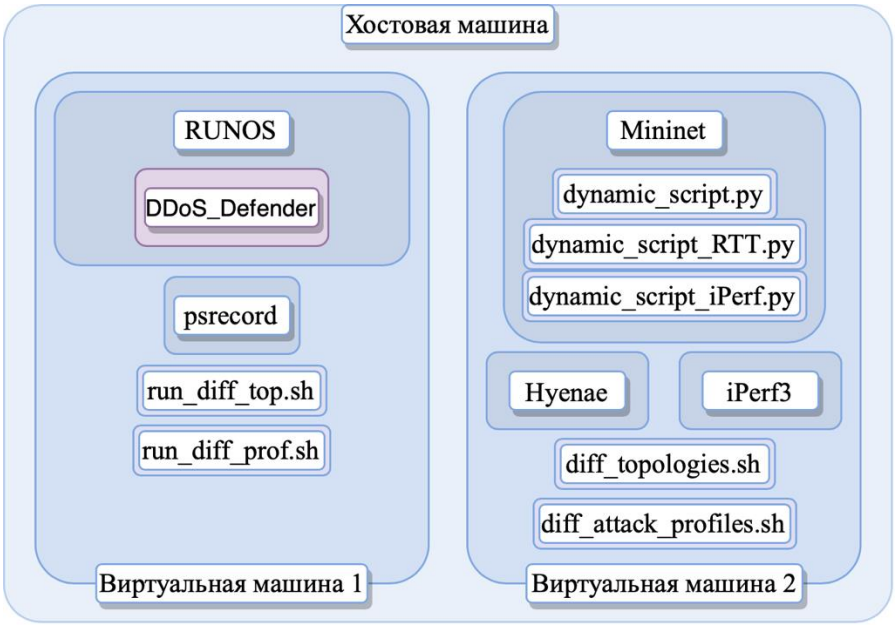

Рис. 12 Схема экспериментального стенда

	Хостовая машина	Машина 1	Машина 2
Операционная система	macOS Catalina 10.15.2 64-bit	Ubuntu 16.04 LTS 32-bit	Ubuntu 18.04 LTS 64-bit
Процессор	Intel Core i5-8210Y CPU 1.60GHz	Intel Core i5-8210Y CPU 1.60GHz	Intel Core i5-8210Y CPU 1.60GHz
Оперативная память	8 Гб	1 Гб	3,8 Гб
IP адрес в локальной сети	-	192.168.56.101	192.168.56.102

Таблица 6 Аппаратные характеристики экспериментального стенда

6.1.2 Исследование качества работы метода

Качество работы метода оценивается по следующим критериям:

1. Оценка задержки обработки пользовательских потоков.
2. Оценка доступной пропускной способности между пользовательскими хостами.
3. Оценка утилизации ЦПУ и ОП контроллера.
4. Оценка использования таблиц потоков в атакуемых коммутаторах.
5. Оценка ошибок 1-го и 2-го рода.

Эта часть исследования проводится на одном типе топологии с 9 хостами (Рис. 13), имеющей 3 зараженных хоста. При этом:

- Хосты могут изменять IP адрес динамически по протоколу DHCP.
- Хосты могут отключаться, подключаться (как те, что изначально были в топологии, так и новые) и мигрировать между граничными коммутаторами.
- Зараженные хосты могут генерировать UDP пакеты с различными IP адресами с частотой 5 – 100 пакетов в секунду.
- Пользовательские хосты генерируют UDP трафик с частотой установки новых потоков 4 – 6 потоков в минуту и количеством 9 – 10 пакетов на поток. Таким образом, частота отправки пакетов с хостов пользователей примерно равна 1 пакет в секунду.
- Коммутаторы могут отключаться и подключаться вновь.

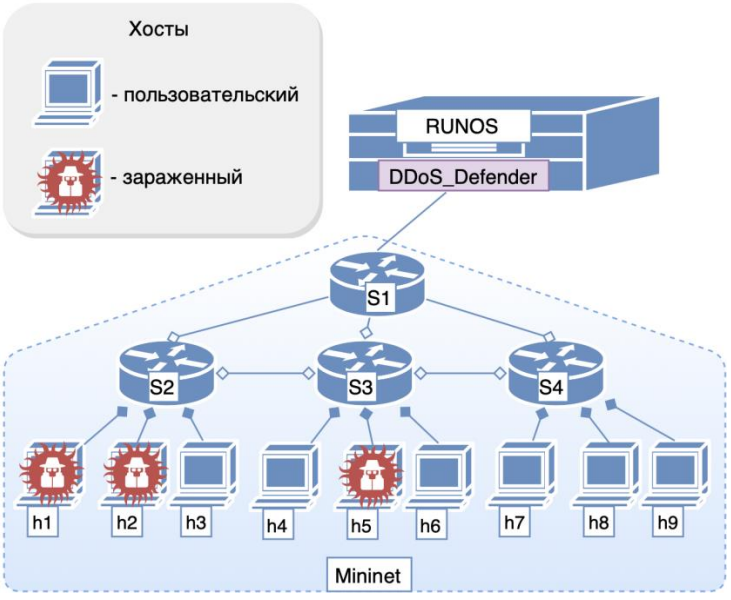

Рис. 13 Экспериментальная топология

Во время эксперимента хосты h_7 и h_8 производят миграцию на коммутаторы S_2 и S_3 соответственно. Хосты h_2 и h_4 сменяют свои IP адреса с помощью DHCP сервера, а хост h_3

– статически. Коммутатор S_1 на время отключается и подключается вновь. Временная диаграмма поведения узлов в топологии представлена на рисунке (Рис. 14). Для воспроизведения динамики в топологии эксперимента написан скрипт для Mininet dynamic_script.py. Чтобы запустить скрипт, необходимо ввести «sudo python3 dynamic_script.py».

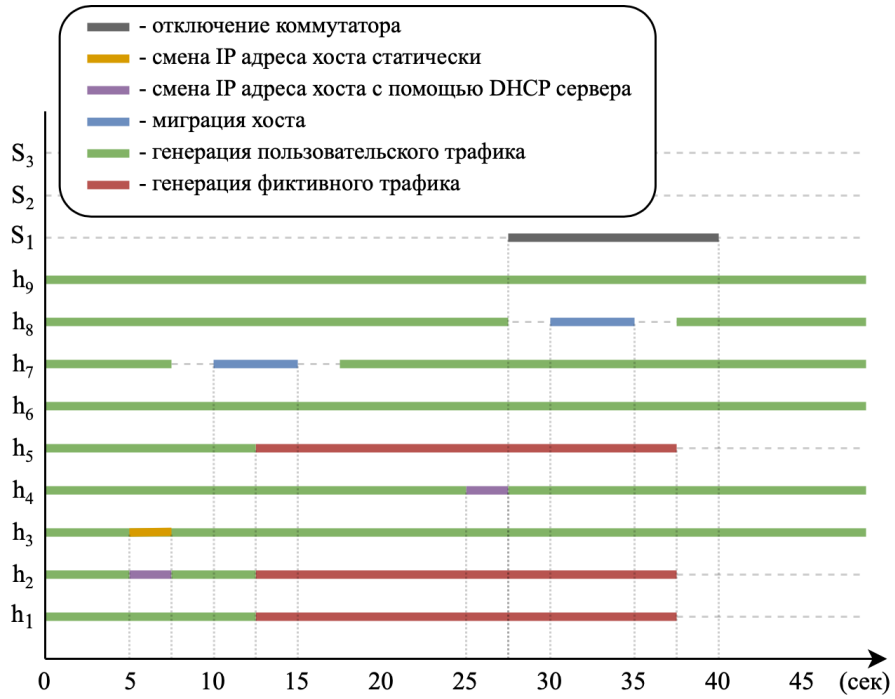

Рис. 14 Диаграмма поведения узлов в топологии для исследования качества работы метода

Проверка 1: оценка задержки обработки пользовательских потоков. Проводятся измерения *RTT* для первого пакета пользовательского потока с помощью *ping*. *RTT* замеряется между хостами h_4-h_6, h_3-h_4 и h_7-h_9 (после миграции h_7 будет подключен к порту коммутатора S_2):

- h_4-h_6 – связь через граничный коммутатор;
- h_3-h_4 – связь через одно соединение между граничными коммутаторами;
- h_7-h_9 – связь через два соединения между граничными коммутаторами;

Для этого производится запуск измененного скрипта dynamic_script_RTT.py, в котором на 17 секунде эксперимента в терминалах хостов h_3, h_4 и h_7 запускается команда ping и полученное значение задержки для первого пакета потока записывается в файл RTT.txt. Гарантируется, что до этого в коммутаторах отсутствовало правило потока между этими парами хостов. Эксперимент проводится 2 раза: 1 раз без использования DDoS_Defender и 1 раз вместе с ним.

Проверка 2: оценка доступной пропускной способности между пользовательскими хостами. Проводятся измерения доступной пропускной способности с помощью *iPerf3*. Пропускная способность измеряется между h_4-h_6 и h_3-h_9. Для воспроизведения эксперимента запускается измененный скрипт dynamic_script_iPerf.py, в котором на 10 секунде эксперимента на хостах h_6, h_9 запускаются серверы iPerf, а на h_4, h_3 – клиенты iPerf. Данные записываются в файл iPerf.txt с 10 по 20 секунды. Эксперимент проводится 2 раза: 1 раз без использования DDoS_Defender и 1 раз вместе с ним.

Проверка 3: оценка утилизации ЦПУ и памяти контроллера. Для проведения эксперимента запускается скрипт dynamic_script.py и на машине 2, на которой запущен контроллер, в терминале запускается команда *«psrecord $(pgrep runos) --interval 1 --plot plot.png --log activity.txt»*. Утилита *psrecord* записывает лог об измерении утилизации ЦПУ в процентах и утилизации памяти в МБ в файл activity.txt и строит график для измеряемых величин. Эксперимент проводится 2 раза: 1 раз без использования DDoS_Defender и 1 раз вместе с ним.

Проверка 4: оценка использования таблиц потоков в атакуемых коммутаторах. В этом эксперименте измеряется количество записей потоков с полем in_port, совпадающим с номерами портов, к которым подключены пользовательские и зараженные хосты. Далее производится расчет доли потоков зараженных хостов от общего количества потоков. Статистика с атакуемых коммутаторов собирается 3 раза (на 4, 20 и 40 секундах) с целью оценить, насколько сильно утилизируется свободное место в таблицах коммутаторах во время атаки с использованием метода и без него – разница в замерах на 4 и 20 секундах, – и насколько качественно предложенный метод справляется с устранением последствий атаки – разница в замерах на 20 и 40 секундах. Для этого в коде *DDoS_Defender.cc* и *DDoS_Defender.hpp* имеется метод *infected_flows_rate()* (считает долю потоков с зараженных хостов и выводит ее в LOG(INFO) и атрибуты map *test_switch_flow,* struct *flow_stat* в комментариях с пометкой *«//FOR TESTING».* После удаления комментирования данных строк в коде приложения необходимо пересобрать контроллер. Теперь при запуске скрипта dynamic_script.py в *LOG(INFO)* контроллера будет выводиться доля потоков с зараженных хостов на каждом атакуемом коммутаторе (Рис. (?)). Эксперимент проводится 2 раза: 1 раз без использования DDoS_Defender и 1 раз вместе с ним.

[Рис. (?) *скрин логов руноса появится после проведения экспериментов*]

Проверка 5: оценка ошибок 1-го и 2-го рода. Под ошибкой 1-го рода *FP* будем понимать долю пользовательских потоков, не обработанных контроллером. Ошибка 2-го рода *FN* – доля фиктивных потоков, обработанных контроллером как пользовательские. Будем рассчитывать ошибки по формулам:

$$FP = \frac{PI_{user} - PO_{user}}{PI_{user}} \tag{6-1}$$

$$FT = \frac{PO_{inf}}{PI_{inf}} \tag{6-2}$$

где PI_{user}, PI_{inf} – количество Packet-In от пользовательских и зараженных хостов, PO_{user}, PO_{inf} – количество Packet-Out от пользовательских и зараженных хостов.

Для проведения эксперимента в коде *DDoS_Defender.cc* и *DDoS_Defender.hpp* имеется метод *compute_FP_FT()* (рассчитывает ошибки 1-го и 2-го рода по формулам (6-1), (6-2) и выводит их значения в LOG(INFO) и атрибуты map *test_hosts_PI_PO,* struct *hosts_stat* в комментариях с пометкой *«//FOR TESTING».* После удаления комментирования данных строк в коде приложения необходимо пересобрать контроллер. Теперь при запуске скрипта dynamic_script.py в *LOG(INFO)* контроллера будут выведены значения ошибок 1-го и 2-го рода после 40-й секунды (Рис. (?)). Эксперимент проводится 2 раза вместе с применением метода.

[Рис. (?) *скрин логов руноса появится после проведения экспериментов*]

6.1.3 Исследование стабильности работы метода

Стабильность работы метода оценивается на основе значений ошибок 1-го и 2-го рода. Данная часть экспериментального исследования проводится на различных профилях активности зараженных хостов и одной топологии, использовавшейся для проведения экспериментов в пункте 6.1.2. Рассматриваются три паттерна изменения мощности DDoS атаки (Рис. 15):

1. Возрастающая мощность. Паттерн имеет характеристику k – коэффициент наклона кривой на графике (Рис. 15). Далее приведены возможная реализация данного профиля активности:
   1.1. Мощность атаки растет по доле активных зараженных хостов (act_rate).
   1.2. Мощность атаки растет по частоте отправки пакетов с зараженных хостов ($send\_freq$).
   1.3. Мощность атаки растет по act_rate и $send\_freq$.

2. Пульсирующая мощность. Паттерн имеет следующие характеристики:
    - att_time – интервал атаки.
    - $wait\_time$ – интервал между атаками (ожидание).
    - att_freq – частота отправки пакетов во время атаки.
    - $wait\_freq$ – частота отправки пакетов во время ожидания.
    - inf_rate – доля активных зараженных хостов, участвующих в проведении атаки.
3. Волнообразная мощность. Паттерн имеет характеристики:
    - T – период синусоидальной кривой
    - A – амплитуда частоты отправки пакетов во время атаки.

    3.1. Изменение мощности за счет доли активных зараженных хостов (act_rate).

    3.2. Изменение мощности за счет частоты отправки пакетов с зараженных хостов ($send\_freq$).

    3.3. Изменение мощности за счет act_rate и $send\_freq$.

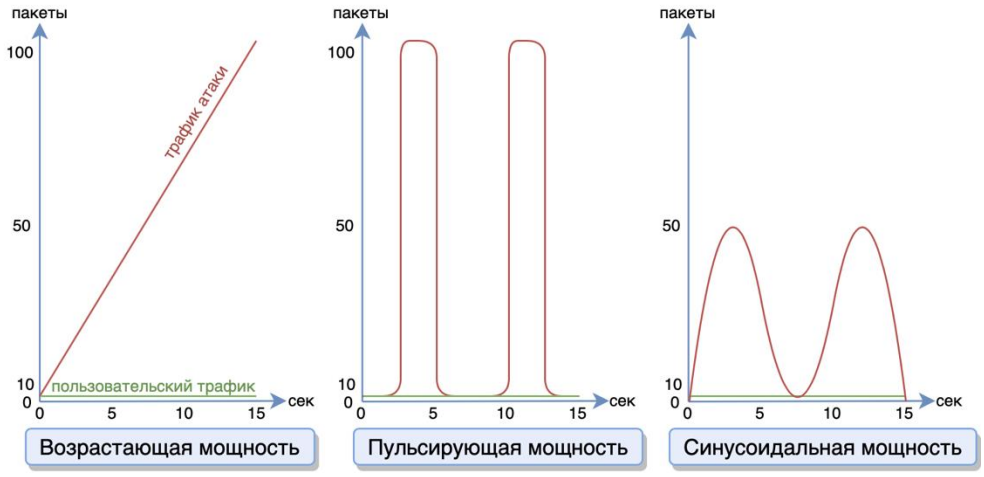

Рис. 15 Паттерны изменения мощности DDoS атаки

В исходную топологию, описанную в скрипте dynamic_script.py, были внесены изменения для возможности проведения атаки с разными профилями активности зараженных хостов, где параметр [1|2|3] указывает, какую из возможных реализаций паттерна атаки необходимо воспроизвести:

- *grow_<k>_<[1|2|3]>*;
- *pulse_<att_time>_<wait_time>_<att_freq>_<wait_freq>_<inf_rate>*;
- *wave_<T>_<A>_<[1|2|3]>*;

Поведение узлов в топологии показано на рисунке (Рис. 14). Эта часть исследования требует как минимум 7 запусков экспериментальной топологии (без изменения характеристик паттернов атак). Чтобы автоматизировать проведения

эксперимента, на машине 2 необходимо запустить скрипт run_diff_prof.sh, который перезапускает контроллер для каждого эксперимента и записывает выведенные в LOG(RUNOS) значения ошибок 1-го и 2-го рода в файл test_stability.txt; на машине 1 – скрипт diff_attack_profiles.sh, который поочередно производит запуски Mininet-скриптов с экспериментальными топологиями с различными паттернами атаки и их реализациями. Перед воспроизведением экспериментов для подсчета и вывода значений ошибок 1-го и 2-го рода необходимо удалить комментирования метода *compute_FP_FT()* в коде DDoS_Defender.hpp и DDoS_Defender.cc и атрибутов map *test_hosts_PI_PO,* struct *hosts_stat.*

6.1.4 Исследование эффективности работы метода

Эффективность работы метода оценивается на основе ошибок 1-го и 2-го рода. Тестирование производится на топологиях с различными характеристиками:

- h_amount – количество хостов (может изменяться во время работы сети).
- inf_rate – максимальная доля зараженных хостов ($0 < inf\_rate < 1$).
- new_freq – частота подключения новых хостов $\left[\frac{\text{количество}}{\text{1 минута}}\right]$.
- mig_freq – частота миграции хостов $\left[\frac{\text{количество}}{\text{1 минута}}\right]$.
- sw_rate – доля граничных коммутаторов, которые могут одновременно отключиться ($0 \leq sw\_rate < 1$).

В исследовании рассматриваются топологии со следующими значениями характеристик:

1. $h\_amount = 9,\ inf\_rate = 0{,}33,\ new\_freq = 5,\ mig\_freq = 15,\ sw\_rate = 0{,}33.$
2. $h\_amount = 9,\ inf\_rate = 0{,}77,\ new\_freq = 5,\ mig\_freq = 15,\ sw\_rate = 0{,}33.$
3. $h\_amount = 27,\ inf\_rate = 0{,}33,\ new\_freq = 5,\ mig\_freq = 15,\ sw\_rate = 0{,}66.$
4. $h\_amount = 64,\ inf\_rate = 0{,}33,\ new\_freq = 20,\ mig\_freq = 30,\ sw\_rate = 0{,}33.$
5. $h\_amount = 128,\ inf\_rate = 0{,}66,\ new\_freq = 5,\ mig\_freq = 15,\ sw\_rate = 0{,}66.$

Для автоматизации проведения эксперимента на машине 2 необходимо запустить скрипт run_diff_topo.sh, который перезапускает контроллер для каждого эксперимента и записывает выведенные в LOG(RUNOS) значения ошибок 1-го и 2-го рода в файл test_efficiency.txt; на машине 1 – скрипт diff_topologies.sh, который поочередно производит запуски Mininet-скриптов с топологиями, приведенными выше. Перед воспроизведением экспериментов для подсчета и вывода значений ошибок 1-го и 2-го рода необходимо удалить комментирования метода *compute_FP_FT()* в коде DDoS_Defender.hpp и DDoS_Defender.cc и атрибутов map *test_hosts_PI_PO,* struct *hosts_stat.*

6.1.5 Исследование оптимального значения интервала сбора статистики метода

В реализации разработанного метода присутствует сбор статистики записей потоков с граничных коммутаторов, который включает в себя отправку запросов с контроллера на каждый граничный коммутатор и получение данных о каждой записи потока на них. Если интервалы между сбором статистики будут достаточно малы, то этот процесс будет нагружать контроллер и утилизировать доступную пропускную способность каналов. С другой стороны, исходя из реализации метода, частый сбор статистики увеличит скорость обнаружения и устранения DDoS атаки. Таким образом, в данной части экспериментального исследования производится оценка скорости обнаружения, противодействия и устранения последствий атаки в зависимости от интервала сбора статистики, а также оценка оптимального значения интервала сбора статистики, которое минимизирует утилизацию ЦПУ контроллера во время противодействия атаке.

Эксперименты проводятся на топологии, описанной в Mininet-скрипте dynamic_script.py и воспроизводящей поведение хостов согласно рисунку (Рис. 14). Скорость обнаружения атаки измеряется как разность между временем начала атаки и временем лога контроллера с текстом «ATTACK START» (Рис. 11). Скорость противодействия атаки – как разность времени лога «ATTACK END» и лога «ATTACK START». Скорость устранения последствий атаки – как разность времени лога «END CLEAR TABLES» и лога «ATTACK END».

Для воспроизведения эксперимента на машине 2 после запуска контроллера в терминале запускается команда *«psrecord $(pgrep runos) --interval 1 --plot plot.png --log*

activity.txt». Утилита *psrecord* записывает лог об измерении утилизации ЦПУ в процентах и утилизации памяти в МБ в файл activity.txt и строит график для измеряемых величин. На машине 1 запускается топология с помощью команды «sudo python3 dynamic_script.py».

6.2 Результаты

6.2.1 Оценка качества работы метода

6.2.2 Оценка стабильности работы метода

6.2.3 Оценка эффективности работы метода

6.2.4 Оценка оптимального значения интервала сбора статистики метода

6.3 Выводы

Список литературы

1. Смелянский Р.Л. Программно-конфигурируемые сети // Открытые системы [Электронный ресурс]. – URL: http://www.osp.ru/os/2012/09/13032491 (дата обращения: 03.04.2017).
2. Open Networking Foundation. Software-Defined Networking: The New Norm for Networks // ONF White Paper. – 2012. – Т. 2. – С. 2-6. - URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf
3. Шаньгин В. Ф. Информационная безопасность компьютерных систем и сетей. – 2008.
4. Encyclopedia by Kaspersky Lab – URL: https://encyclopedia.kaspersky.ru/glossary/dos-denial-of-service-attack/
5. Encyclopedia by Kaspersky Lab – URL: https://encyclopedia.kaspersky.ru/glossary/ddos-distributed-denial-of-service-attack/
6. Open Networking Foundation. OpenFlow Switch Specification. Version 1.3.0 (Protocol version 0x04); June 25, 2012; ONF TS-025 – URL: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf
7. Kalkan K., Gur G., Alagoz F. Defense mechanisms against DDoS attacks in SDN environment //IEEE Communications Magazine. – 2017. – Т. 55. – №. 9. – С. 175-179.
8. Lim S. et al. Controller Scheduling for Continued SDN Operation under DDoS Attacks //Electronics Letters. – 2015. – Т. 51. – №. 16. – С. 1259-1261.
9. Wang S., Chavez K. G., Kandeepan S. SECO: SDN sEcure COntroller algorithm for detecting and defending denial of service attacks //2017 5th International Conference on Information and Communication Technology (ICoIC7). – IEEE, 2017. – С. 1-6.
10. Kalkan K. et al. JESS: Joint Entropy-Based DDoS Defense Scheme in SDN //IEEE Journal on Selected Areas in Communications. – 2018. – Т. 36. – №. 10. – С. 2358-2372.

11. Peng T., Leckie C., Ramamohanarao K. Protection from distributed denial of service attacks using history-based IP filtering //IEEE International Conference on Communications, 2003. ICC'03. – IEEE, 2003. – T. 1. – C. 482-486.
12. Mininet homepage [HTML] (https://github.com/mininet/mininet).
13. RUNOS v2.0 homepage [HTML] (https://github.com/ARCCN/runos).
14. Hyenae 0.36-1 homepage [HTML] (https://sourceforge.net/projects/hyenae/).
15. iPerf3 homepage [HTML] (https://github.com/esnet/iperf).
16. Psrecord homepage [HTML] (https://github.com/astrofrog/psrecord).
17. Oracle VM VirtualBOX 6.0 [HTML] (https://www.virtualbox.org).
18. DDoS_Defender https://github.com/anyaantipina/DDoS_Defender_1.0