Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

**Институт компьютерных наук и технологий**

**Кафедра Компьютерные интеллектуальные технологии**

**ОТЧЁТ**

**О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ**

**на тему: Проектирование высокоуровневой части IoT устройства для отслеживания изменений в сети Интернет**

направление подготовки: Прикладная информатика

Выполнил(а):

студент группы 33506/3

Киселёв Валентин Николаевич

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель:

Подпись\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка «\_\_\_\_\_\_\_\_\_\_\_\_\_\_»

Санкт-Петербург

2018

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc515467471)

[Введение 3](#_Toc515467472)

[Актуальность сферы IoT 3](#_Toc515467473)

[Цели и задачи исследовательской работы 4](#_Toc515467474)

[1. Анализ актуальных IoT технологий 4](#_Toc515467475)

[1.1. Рассматриваемые устройства 4](#_Toc515467476)

[1.2. Критерии сравнения 5](#_Toc515467477)

[2. Проектирование IoT устройства 6](#_Toc515467478)

[2.1. Декомпозиция устройства 6](#_Toc515467479)

[2.2. Декомпозиция слушателя 7](#_Toc515467480)

[3. Реализация IoT устройства 9](#_Toc515467481)

[3.1. Протокол сообщения 9](#_Toc515467482)

[3.2. Слушатель 9](#_Toc515467483)

[3.3. Демон 10](#_Toc515467484)

[3.4. Подсистема уведомлений 11](#_Toc515467485)

[3.5. Конфигурирование подсистем 11](#_Toc515467486)

[3.6. Поток исполнения и принцип работы 12](#_Toc515467487)

[ЗАКЛЮЧЕНИЕ 13](#_Toc515467488)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 14](#_Toc515467489)

# Введение

## Актуальность сферы IoT

Интернет вещей (англ. Internet of Things — IoT) – это новый этап эволюционного развития Интернета, значительно расширяющий возможности сбора, анализа и распространения данных. Поскольку прогресс человеческого общества во многом зависит от превращения исходных данных в полезную информацию, и, в конечном итоге, в мудрость. В этом смысле Интернет вещей приобретает огромное значение.

Уже есть проекты, наглядно демонстрирующие его способность преодолеть разрыв между богатыми и бедными, предоставить мировые ресурсы тем, кто больше всего в них нуждается, и помочь нам лучше понять свою планету, чтобы научиться предупреждать проблемы [2]. Вместе с тем есть факторы, замедляющие развитие Интернета вещей. К ним относятся переход к протоколу IPv6, принятие единого набора общих стандартов и разработка источников питания для миллионов (и даже миллиардов) миниатюрных датчиков.

По оценкам компании Ericsson, в 2018 году число датчиков и устройств Internet of Things превысит количество мобильных телефонов и станет самой большой категорией подключенных устройств. Аналитики компании прогнозирует, что из приблизительно 28 млрд подключенных устройств по всему миру, к 2021 году, около 16 миллиардов будут связаны с IoT [1].

Сегодня Интернет вещей подходит к этапу, на котором разнородным сетям и множеству датчиков предстоит объединиться для взаимодействия под управлением единых стандартов. Эта цель требует от коммерческих организаций, государственных учреждений, стандартообразующих органов и учебных заведений общих усилий, направленных на достижение единой цели. Чтобы Интернет вещей обрел популярность у обычных пользователей, поставщики услуг и другие участники рынка должны разработать приложения, значительно повышающие качество жизни простых граждан.

## Цели и задачи исследовательской работы

В рамках моей исследовательской работы я намерен изучить области проектирования и разработки IoT устройств. В связи с этим я выделил следующие задачи:

* выявить наиболее оптимальные и применимые технологии и устройства, которые позволяют создавать функционально небольшие системы для решения узкого круга задач,
* спроектировать и реализовать IoT устройство на основе стека выявленных технологиях, которое сможет отслеживать изменения в сети Интернет и реагировать на них
* проанализировать полученную информационную систему на предмет безопасности, отказоустойчивости и предложить дальнейшие варианты развития.

# 1. Анализ актуальных IoT технологий

Для того, чтобы понять, какая конфигурация IoT устройства будет приемлима для реализации поставленных целей, нужно выбрать наиболее перспективные технологии, определить критерии выбора и с помощью сравнительного анализа выбрать наиболее применимый по всем критериям вариант.

Основой любой информационной системы являются компьютеры и связи между ними. В моём случае IoT устройство также является полноценным копьютером небольшого размера со значительными ограничениями в производительности и вычислительной мощности, которых, тем не менее, достаточно для выполнения поставленных задач.

## 1.1. Рассматриваемые устройства

Среди доступных на рынке устройств присутствуют [3]:

1. Raspberry PI 3 B – новейшая модель небольшого портативного одноплатного компьютера с 1Гб ОЗУ, встроенным адаптером WiFi, 64-битным ARM-процессором Cortex-A53 с тактовой частотой 1,2 ГГц на однокристальном чипе Broadcom BCM2837, 40 контактами GPIO;
2. BeagleBone Black – бюджетный портативный комьютер с 512 Мб ОЗУ, [ARM-процессором Cortex-A8](https://www.ti.com/product/am3358) с тактовой частотой 1ГГц;
3. Radxa Rock2 Square – портативный компьютер с процессором ARM Cortex-A17 с тактовой частотой 1.8Ггц, ОЗУ можно выбрать: 1, 2 или 4 Гб;
4. NanoPC-T1 – портативный одноплатный компьютер на базе ARM-процессора Cortex-A9 Quad-core с тактовой частотой 1.5 Ггц. Имеет аналогичный Raspberry Pi характеристики, но также способен запускать ОС Android 5.0.

Существует множество других устройств, но эти четыре наиболее распространены и используются в сфере IoT. Рассмотрим критерии сравнения этих устройств:

## 1.2. Критерии сравнения

1. Цена. Стоимость устройства должна соответствовать целям и задачам, которые будут на него возложены;
2. Производительность. Устройство должно справляться с нагрузкой;
3. Доступность и документированность. Легче и надёжнее использовать проверенное людьми решение, которое используется в рабочих проектах, о котором собрана статистика использования;
4. Количество поддерживаемых типов интерфейсов ввода-вывода. Так как устройство должно функционировать в сети интернет, а также визуализировать входящие сообщения, оно должно обладать некоторым набором портов ввода-вывода.

Так как целью моей работы стало проектирование устройтва, я остановился на портативном компьютере Raspberry Pi 3 B. Из прочих оно выделяется невысокой ценой, а также поддержкой интерфейсов: GPIO (низкоуровневые порты ввода-вывода), HDMI, Ethernet, USB. Кроме того, для Raspberry Pi существует большой выбор операционных систем, оптимизированных специально под это устройство. Среди них как UNIX, так и Windows системы.

Для данного проекта наиболее оптимальным выбором операционной системы для меня стал дистрибутив Linux Arch для ARMv8 архитектуры. Среди кандидатов были Raspbian, CentOS Core, FreeBSD 11, Alpine Linux. К сожалению размеры Raspbian и CentOS были значительно больше остальных (3 Гб против 700Мб), FreeBSD 11 поддерживала не все порты ввода-вывода, а Alpine оказалась сложна для настройки и конфигурирования, так как требовала наличия монитора и подключения по HDMI. Дистрибутив Arch Linux славится своей универсальностью, надёжность и большим сообществом, поэтому идеально подходит в роли операционной системы для IoT устройства.

# 2. Проектирование IoT устройства

Микросервисная архитектура в настоящее время становится наиболее применимой в решениях современных задач разработки информационных систем. Практическая реализация мало связанных подсистем может потребовать больше усилий и планирования на начальном этапе, однако такой подход оправдывает себя в будущем.

## 2.1. Декомпозиция устройства

Для того, чтобы понять, на какие подсистемы нужно разделить данный проект, рассмотрим функциональные требования, поставленные ранее:

* устройство должно принимать уведомления из сети;
* устройство должно иметь возможность поддерживать новые технологии коммуникации;
* устройство должно сигнализировать о новых событиях, к примеру мигать светодиодом.

Итак, как минимум три основные сферы обязанности можно смело определить. Если первые две относятся исключительно к программной реализации, то для третьего требования необходим модуль сигнализации. Мы с Андреем пришли к выводу, что это устройство необходимо сделать автономным, независимым от устройств, сигнализирующих о получении сообщения. Таким образом, получится сузить область ответственности, хотя и придётся описывать способ общения между устройством приёма и отображения событий.

Реализацией устройства, отвечающего за сигнализацию о входящих событиях, занялся Андрей. Моей задачей стала реализация устройства, выполняющего роль приёмника сообщений. Для простоты я ограничился реализацией протокола IMAP, чтобы получать новую почту и оповещать об этом устройства сигнализирования. Таким образом, устройство, принимающее уведомления становится высокоуровневой частью IoT системы, а устройство, сигнализирующее об этом уведомлении становится низкоуровневой частью проекта.

## 2.2. Декомпозиция слушателя

Высокоуровневое устройство должно слушать (мониторить) события в сети интернет, вызываемые почтой, социальными сетями или чем-то ещё. Так или иначе должен быть источник, у которого точно можно узнать, что что-то произошло. В качестве примера я рассматриваю сообщения электронной почты. Напрашивается абстрагирование уведомления до некоего формата, а также абстрагирования сервиса, который будет изучать сеть на предмет новых уведомлений.

Абстрагировать уведомление можно, например, оформив его в протокол, сообщение определённой длины, которое можно будет передавать низкоуровневой части устройства.

Абстрагировать сервис получения уведомлений – высокоуровневая задача. Для её реализации необходимо написать код на языке программирования и определить интерфейс, который должны реализовывать новые типы слушателей.

Выбор языка программирования – важная часть в этой задаче, поскольку язык программирования гарантирует или не гарантирует простоту, ясность и возможность реализации намеченных задач. Выбор был между языками:

* Python 3.6
* Ruby 2.3
* Go 1.9

В силу специфики форматирования кода, широкой распространённости и большого количества готовых библиотек был выбран язык Python версии 3.6 (самой новой на данный момент).

Интерфейс слушателя (или монитора) был выбран самый простой. Любой слушатель должен обладать одним методом, возвращающим количество новых уведомлений. Это будет кстати при развитии проекта, когда, вероятно, нужно будет сигнализировать не только о получении новых уведомлений, но и об их количестве.

Так как слушатель должен работать в фоне, необходима реализация фоновой программы, которая будет запускаться и переходить в фоновый процесс. Для этого наилучший способ – реализовать демона. Демон – в рамках операционных систем Linux – это программа, работающая в фоне. А значит, нужно выявить требования к реализации демона, чтобы сделать демона корректно.

Помимо этого, мы можем предположить, что протокол общения с сигнализирующим устройством изменится. А возможно поменяется полностью подход и принцип функционирования устройства. Например – мы захотим направлять уведомления на почту. Для того, чтобы сделать такое возможным стоит предусмотреть абстрацию типа «уведомитель». Этот интерфейс будет требовать реализации лишь одного метода: уведомить.

Так как устройство будет использовать пользовательские файлы, необходимо реализовать абстрактный модуль конфигурирования, который будет предоставлять демону необходимые данные для подключения к сервисам уведомлений и к сигнализирующему устройству. На практике эти данные могут храниться в файле, в базе данных, на другом устройстве. Но такой модуль должен составлять конфигурацию и предоставлять интерфейс для получения нужных значений.

Все подсистемы слушателя были описаны, был выбран подход максимизировать уровень абстракций, чтобы все эти подсистемы можно было разрабатывать и тестировать независимо. Поскольку объём работы для каждой из этих подсистем небольшой, это будет гораздо проще разработки монолитного приложения с трудно заменяемыми или не заменяемыми вовсе частями.

# 3. Реализация IoT устройства

Рассмотрим реализацию описанных подсистем, рассмотренных во второй главе.

## 3.1. Протокол сообщения

Подробно этот протокол описан в работе Андрея, поэтому я приведу только базовые описания полей протокола:

* Head – сигнализирует о начале полезных данных, 8 байт,
* Sender – код отправителя (источник) события, 8 байт,
* Type – идентификатор (тип) события, 4 байта,
* Value – значение (параметры) события, 12 байт,
* Check-sum – контрольная сумма кадра, 8 байт,
* Ending – концевик кадра, 8 байт;

Важная часть в протоколе – поля Sender, Type, Value. Они должны кодироваться в высокоуровневой части устройства и декодироваться в низкоуровневой части. К сожалению, мы имеем не полностью разработанную и нацеленную на конкретный тип уведомлений систему. Поэтому дальнейшая реализация способа отображения каждого типа сообщения, источника и количества, закодированный числами, остаётся неопределённой. Но это вопрос реализации, а не архитектуры.

## 3.2. Слушатель

В качестве примера реализации был выбран протокол IMAP для получения новых сообщений из почтового ящика. Конфигурация требует предоставления адреса электронной почты и пароля для доступа. Метод получения количества сообщений возвращает количество непрочтённых новых сообщений. Используется библиотека Python imaplib, имеющая необходимые классы для реализации таких действий. В ходе реализации были также созданы специальные исключения, которые вызываются слушателем, если аутентификация не удалась. Задача обработки этих исключения лежит на подсистеме демона.

## 3.3. Демон

Программа-демон должна удовлетворять некоторым требованиям [4]:

* демон должен запускаться лишь один раз, двух одинаковых демонов быть не должно,
* демон не должен быть ассоциирован со стандартным потоком ввода-вывода,
* демон не должен быть дочерним процессом какой-либо пользовательской программы, например, эмулятора терминала,
* демон должен записывать в лог-файлы информативные сообщения;

Логика работы демона может меняться, однако код, позволяющий выполнить приведённые условия всегда остаётся. Поэтому я решил сделать абстрактный класс Daemon, который обеспечивает необходимые условия и проверки при старте и запускает метод run, который описывается в реализации необходимым образом.

Для того, чтобы сделать программу независимой от запускающей программы используется метод «двойного ответвления» [5], суть которого – создать дочерний процесс, родительский процесс завершить и повторить действие с дочерним процессом. Таким образом достигается уверенность в том, что процесс становится демоном.

Для правильного функционирования от процесса «отсоединяется» окружение вызвавшей стороны, изменяется текущая директория. Очищаются буферы стандартных потоков вывода и ошибок. Происходит перенаправление стандартных потоков ввода, вывода и ошибок на специальное последовательное устройство /dev/null.

Процесс записывает свой PID в файл, который позволяет ему контролировать, был ли он запущен ранее, чтобы не запустить повторно ещё один демон.

Процесс получает при инициализации объект config, в котором содержится путь до директории, куда нужно записывать файлы логов. Уровни логирования и названия файлов нужно указывать в том же классе, в классе абстрактного демона.

## 3.4. Подсистема уведомлений

В нашем проекте мы производим уведомление по serial-интерфейсу напрямую от Raspberry Pi к Arduino. Это один из самых простых способов коммуникации. В открытом доступе существует библиотека serial для Python [6], которая позволяет легко настроить передачу. Но serial-интерфейс – далеко не единственный из возможных вариантов уведомления. Можно передавать данные через Ethernet и Wi-Fi с помощью специальных адаптеров.

Так как мы решили максимально абстрагировать подсистемы, то напрашивается решение определить абстрактый интерфейс уведомления, содержащий только один метод notify, принимающий сообщение. Сообщение – это структура, содержащая поля заголовков протокола коммуникации, описанного выше и более детально разобранного в работе Андрея. Определим ответственность за составление структуры сообщения на подсистему слушателя.

Для тестирования подсистемы я создал реализацию интерфейса, записывающую новые сообщения в файл. Позже заменив его реализацией, работающей с последовательным портом, мне нужно было лишь изменить название класса при создании объекта уведомителя в демоне. Таким образом у кода возросла читабельность, а систему можно было тестировать независимыми частями.

## 3.5. Конфигурирование подсистем

Конфигурирование подсистем стало сложной задачей, ведь все подсистемы были независимыми, но должны были централизованно конфигурироваться. Для этих целей я использовал библиотеку configparser, поставляемую с Python по умолчанию. Для конфигурации использовался один файл с ini-подобным типом записи параметров.

Был создан класс, считывающий конфигурационный файл и реализующий интерфейс словаря, отчего стало возможно обращаться с объектом класса, как со структурой.

## 3.6. Поток исполнения и принцип работы

В коде демона во время инициализации и произведения необходимых действий для создания программы-демона, считывается файл конфигурации с помощью вспомогательного класса. Далее запускается метод run, который явно описывает класс-реализация. В этом файле в зависимости от конфигурации могут быть предприняты действия, настраивающие объекты слушателей и уведомителей. Потом ожидается приём новых сообщений от слушателей, а в случае приёма вызывается метод notify у уведомителя.

Такая прозрачная и декомпозированная архитектура проекта позволила сузить области, в которых возможны появления ошибок, а также намного упростила разработку и тестирование программных частей.

# ЗАКЛЮЧЕНИЕ

В ходе исследования мы заключили следующие выводы:

* современные технологии позволяют создавать IoT устройства, используя известные методы коммуникаций быстро и легко,
* декомпозиция архитектуры на малосвязанные блоки позволяет ускорить и улучшить процесс проектирования,
* использование демонов в операционных системах семейства Linux позволяет создавать системы, работающие со множествами задач эффективно,
* использование связки устройство с ОС Linux + Arduino является правильным декомпозиционным решением, поскольку никак не связывает два устройства ни общим питанием, ни способом общения,
* применение собственных прикладных протоколов общения между устройствами позволяет разрабатывать эти устройства независимо, что ускоряет разработку и упрощает процесс нахождения ошибок.

Таким образом, современные технологии IoT принципиально не отличаются от хорошо спроектированных информационных систем. Единственное важное отличие – создание системы, решающей узкий круг задач. Дальнейшая работа возможна в области безопасности этих устройств, что скорее всего приведёт к усложнению протокола и принципа общения между устройствами.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Интернет вещей // WIKIPEDIA.ORG: Википедия: Свободная энциклопедия – URL: https://ru.wikipedia.org/wiki/Интернет\_вещей – (дата обращения 23.05.2018)
2. Интернет вещей. Как изменится вся наша жизнь на очередном витке развития Всемирной сети. Дейв Эванс // CISCO.COM: Официальный сайт компании CISCO – URL: https://www.cisco.com/c/dam/global/ ru\_ru/assets/executives/pdf/internet\_of\_things\_iot\_ibsg\_0411final.pdf – (дата обращения 23.05.2018)
3. 10 Best Raspberry Pi and Pi 2 Alternatives // BEEDOM.COM: It News – URL: <https://beebom.com/raspberry-pi-and-pi-2-alternatives/> – (дата обращения: 20.05.2018)
4. Linud Daemon Writing HOWTO // [NETZMAFIA.DE: Хакеры в мире Linux – URL: /](netzmafia.de/)<http://www.netzmafia.de/skripten/unix/linux-daemon-howto.html> – (дата обращения: 23.05.2018)
5. Python demonizer class // GITHUB.COM: GitHub: A development platform – URL: <https://github.com/serverdensity/python-daemon> - (дата обращения: 8.04.2018)
6. Docs Pyserial // PYSERIAL.READTHEDOCS.IO: Сайт с документацией к библиотеке Python pyserial – URL: <http://pyserial.readthedocs.io/en/latest/pyserial.html> - (дата обращения: 18.05.2018)

# ПРИЛОЖЕНИЯ

## Приложение 1. Текст класса абстрактного демона

import sys

import os

import time

import atexit

import signal

import logging

from config import Config

class Daemon:

"""A generic daemon class.

Usage: subclass the daemon class and override the run() method."""

def \_\_init\_\_(self, pidfile):

self.pidfile = pidfile

self.config = Config()

logging.basicConfig(filename = self.config['logging']['path'] + '/notitier\_daemon\_log.log',

filemode="w",

level = self.config['logging']['level'],

format = '%(asctime)s %(levelname)s: %(message)s',

datefmt = '%Y-%m-%d %I:%M:%S')

logging.debug("Daemon initialized.")

def \_daemonize(self):

"""Deamonize class. UNIX double fork mechanism."""

try:

# копирует текущий процесс и возвращает 0,

# если теперь находимся внутри дочернего

# процесса, и PID внутри родительского

pid = os.fork()

if pid > 0:

# exit first parent

sys.exit(0)

except OSError as err:

logging.error('fork #1 failed: {0}\n'.format(err))

sys.exit(1)

# decouple from parent environment

os.chdir('/')

os.setsid()

os.umask(0)

# do second fork

try:

pid = os.fork()

if pid > 0:

# exit from second parent

sys.exit(0)

except OSError as err:

logging.error('fork #2 failed: {0}\n'.format(err))

sys.exit(1)

# redirect standard file descriptors

sys.stdout.flush()

sys.stderr.flush()

si = open(os.devnull, 'r')

so = open(os.devnull, 'a+')

se = open(os.devnull, 'a+')

os.dup2(si.fileno(), sys.stdin.fileno())

os.dup2(so.fileno(), sys.stdout.fileno())

os.dup2(se.fileno(), sys.stderr.fileno())

# write pidfile

atexit.register(self.\_delpid)

pid = str(os.getpid())

with open(self.pidfile,'w+') as f:

f.write(pid + '\n')

def \_delpid(self):

os.remove(self.pidfile)

def start(self):

"""Start the daemon."""

# Check for a pidfile to see if the daemon already runs

try:

with open(self.pidfile,'r') as pf:

pid = int(pf.read().strip())

except IOError:

pid = None

if pid:

message = "pidfile {0} already exist. " + \

"Daemon already running?\n"

logging.error(message.format(self.pidfile))

sys.exit(1)

# Start the daemon

self.\_daemonize()

logging.debug("Daemon started")

self.\_run()

def stop(self):

"""Stop the daemon."""

# Get the pid from the pidfile

try:

with open(self.pidfile,'r') as pf:

pid = int(pf.read().strip())

except IOError:

pid = None

if not pid:

message = "pidfile {0} does not exist. " + \

"Daemon not running?\n"

logging.warning(message.format(self.pidfile))

return # not an error in a restart

# Try killing the daemon process

try:

while 1:

os.kill(pid, signal.SIGTERM)

time.sleep(0.1)

except OSError as err:

e = str(err.args)

if e.find("No such process") > 0:

if os.path.exists(self.pidfile):

os.remove(self.pidfile)

else:

logging.error(str(err.args))

sys.exit(1)

logging.debug("Daemon stoped")

def restart(self):

"""Restart the daemon."""

self.stop()

self.start()

def \_run(self):

"""You should override this method when you subclass Daemon. It will be called after the process has been daemonized by start() or restart()."""

raise NotImplementedError("Should implement your own run() realization!")

## Приложение 2. Текст программы демона

import sys

import time

import logging

sys.path.append("./monitor/imap")

sys.path.append("./notifier")

from daemon import Daemon

from config import Config

from imap import Imap, ImapException

from notifier.file import FileNotifier

class NotifierDaemon(Daemon):

def \_run(self):

monitors = list()

out = False

# Init all monitors

if self.config['mail']['enabled']:

try:

imap = Imap(self.config['mail']['login'], self.config['mail']['password'])

except ImapException as e:

logging.error(e)

else:

monitors.append(imap)

logging.debug("NotifierDaemon: All monitors initialized succesful")

# Init notifier

try:

if self.config['notifier']['type'] == "file":

out = FileNotifier()

logging.debug("NotifierDaemon: FileNotifier initialized succesful")

elif self.config['notifier']['type'] == "serial":

out = SerialNotifier()

logging.debug("NotifierDaemon: SerialNotifier initialized succesful")

except Exception as e:

logging.error("NotifierDaemon can't notify ({0})".format(e))

while True:

for monitor in monitors:

# Try get updates from monitors

try:

count = str(monitor.messages\_count())

except ImapException as e:

logging.error(e)

else:

logging.debug("get new messages count from " + type(monitor).\_\_name\_\_ + " = " + count)

# Notifying about updates

try:

out.notify(count)

except Exception as e:

logging.error(e)

time.sleep(self.config['daemon']['timeout'])

## Приложение 3. Пример файла конфигурации

[daemon]

timeout: 5

[logging]

path: .

level: 10

[notifier]

type: serial

[imap]

login: vjy1978577oleg@mail.ru

password: nKxm3uhq8

[serial]

port: 0

boundrate: 9600