Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Центр заочної форми навчання

Кафедра ПІ

Контрольна робота

з дисципліни «Архітектура та проектування ПЗ»

Виконав: ст. гр. ПІз-11-1  Євсеєва Н.С.

Перевірив:                                        к.т.н., доц. каф ПІ Лещинський В. О.

Харків 2015

**Паттерн**

Шаблон проектирования или паттерн (англ. design pattern) в разработке программного обеспечения — повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

Рассмотрим два шаблона проектирования: компоновщик и строитель.

**Шаблон проектирования Компоновщик (Composite)**

Компоновщик (англ. Composite pattern) — шаблон проектирования, объединяет объекты в древовидную структуру для представления иерархии от частного к целому (см. рис1). Компоновщик позволяет клиентам обращаться к отдельным объектам и к группам объектов одинаково.

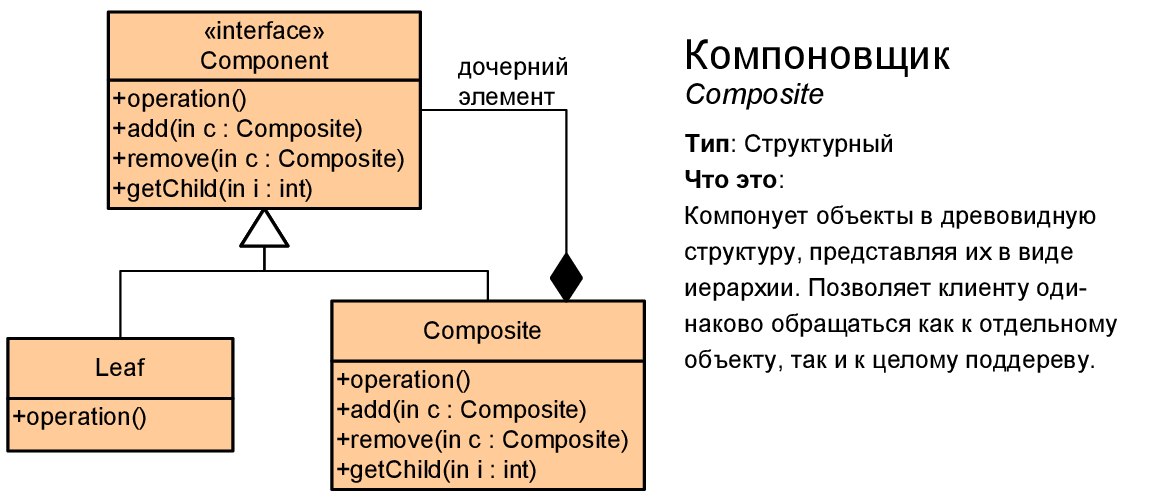


Рисунок 1

Цель

Паттерн определяет иерархию классов, которые состоят из примитивных и сложных объектов, упрощает архитектуру клиента, делает процесс добавления новых видов объекта более простым.

Используют если:

* Необходимо объединять группы схожих объектов и управлять ими.
* Объекты могут быть как примитивными (элементарными), так и составными (сложными). Составной объект может включать в себя коллекции других объектов, образуя сложные древовидные структуры. Пример: директория файловой системы состоит из элементов, каждый их которых также может быть директорией.
* Код клиента работает с примитивными и составными объектами единообразно.

Описание

Паттерн Composite предлагает следующее решение. Он вводит абстрактный базовый класс Component с поведением, общим для всех примитивных и составных объектов. Для случая стратегической игры - это метод getStrength() для подсчета разрушающей силы. Подклассы Primitive and Composite являются производными от класса Component. Составной объект Composite хранит компоненты-потомки абстрактного типа Component, каждый из которых может быть также Composite.

Достоинства паттерна Composite

В систему легко добавлять новые примитивные или составные объекты, так как паттерн Composite использует общий базовый класс Component.

Код клиента имеет простую структуру – примитивные и составные объекты обрабатываются одинаковым образом.

Паттерн Composite позволяет легко обойти все узлы древовидной структуры

Недостатки паттерна Composite

Неудобно осуществить запрет на добавление в составной объект Composite объектов определенных типов. Так, например, в состав римской армии не могут входить боевые слоны.

Пример

#include <iostream>

#include <vector>

#include <assert.h>

 // Component

class Unit

{

  public:

    virtual int getStrength() = 0;

    virtual void addUnit(Unit\* p) {

      assert( false);

    }

    virtual ~Unit() {}

};

 // Primitives

class Archer: public Unit

{

  public:

    virtual int getStrength() {

      return 1;

    }

};

class Infantryman: public Unit

{

  public:

    virtual int getStrength() {

      return 2;

    }

};

 class Horseman: public Unit

{

  public:

    virtual int getStrength() {

      return 3;

    }

};

  // Composite

class CompositeUnit: public Unit

{

  public:

    int getStrength() {

      int total = 0;

      for(int i=0; i<c.size(); ++i)

        total += c[i]->getStrength();

      return total;

    }

    void addUnit(Unit\* p) {

        c.push\_back( p);

    }

    ~CompositeUnit() {

      for(int i=0; i<c.size(); ++i)

        delete c[i];

    }

  private:

   std::vector<Unit\*> c;

};

 // Вспомогательная функция для создания легиона

CompositeUnit\* createLegion()

{

  // Римский легион содержит:

  CompositeUnit\* legion = new CompositeUnit;

  // 3000 тяжелых пехотинцев

  for (int i=0; i<3000; ++i)

    legion->addUnit(new Infantryman);

  // 1200 легких пехотинцев

  for (int i=0; i<1200; ++i)

    legion->addUnit(new Archer);

  // 300 всадников

  for (int i=0; i<300; ++i)

    legion->addUnit(new Horseman);

   return legion;

}

 int main()

{

  // Римская армия состоит из 4-х легионов

  CompositeUnit\* army = new CompositeUnit;

  for (int i=0; i<4; ++i)

    army->addUnit( createLegion());

  cout << "Roman army damaging strength is "

       << army->getStrength() << endl;

  // …

  delete army;

  return 0;

}

## Шаблон проектирования Строитель (Builder)

Паттерн Builder отделяет алгоритм поэтапного конструирования сложного продукта (объекта) от его внешнего представления так, что с помощью одного и того же алгоритма можно получать разные представления этого продукта.

Описание паттерна Builder

Поэтапное создание продукта означает его построение по частям. После того как построена последняя часть, продукт можно использовать.

Для этого паттерн Builder определяет алгоритм поэтапного создания продукта в специальном классе Director (распорядитель), а ответственность за координацию процесса сборки отдельных частей продукта возлагает на иерархию классов Builder. В этой иерархии базовый класс Builder объявляет интерфейсы для построения отдельных частей продукта, а соответствующие подклассы ConcreteBuilder их реализуют подходящим образом, например, создают или получают нужные ресурсы, сохраняют промежуточные результаты, контролируют результаты выполнения операций(см. рис 2).

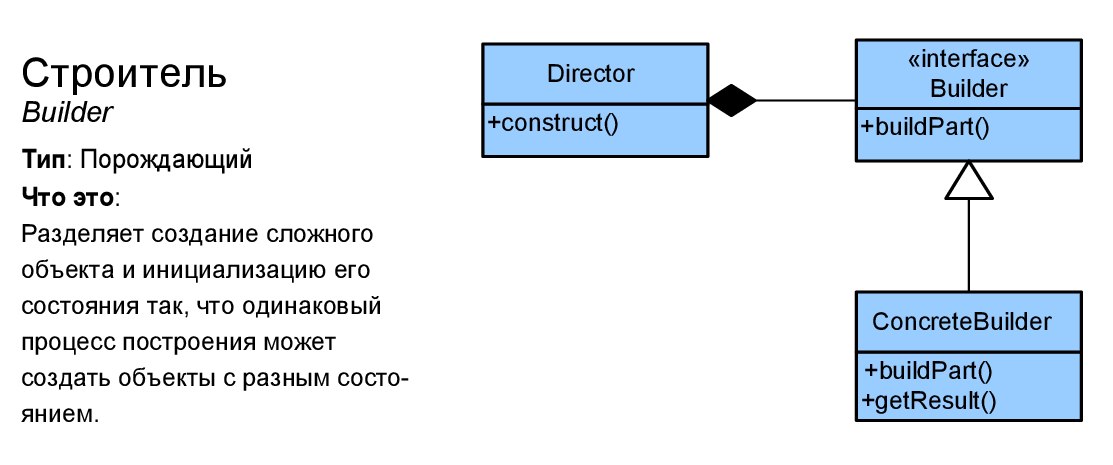


Рисунок 2

Достоинства паттерна Builder

* Возможность контролировать процесс создания сложного продукта.
* Возможность получения разных представлений некоторых данных.

Недостатки

* позволяет изменять внутреннее представление продукта;
* изолирует код, реализующий конструирование и представление;
* дает более тонкий контроль над процессом конструирования.

Применение

* алгоритм создания сложного объекта не должен зависеть от того, из каких частей состоит объект и как они стыкуются между собой;
* процесс конструирования должен обеспечивать различные представления конструируемого объекта.

Пример

#include <iostream>

#include <vector>

// Классы всех возможных родов войск

class Infantryman

{

public:

void info() {

cout << "Infantryman" << endl;

}

};

class Archer

{

public:

void info() {

cout << "Archer" << endl;

}

};

class Horseman

{

public:

void info() {

cout << "Horseman" << endl;

}

};

class Catapult

{

public:

void info() {

cout << "Catapult" << endl;

}

};

class Elephant

{

public:

void info() {

cout << "Elephant" << endl;

}

};

// Класс "Армия", содержащий все типы боевых единиц

class Army

{

public:

vector<Infantryman> vi;

vector<Archer> va;

vector<Horseman> vh;

vector<Catapult> vc;

vector<Elephant> ve;

void info() {

int i;

for(i=0; i<vi.size(); ++i) vi[i].info();

for(i=0; i<va.size(); ++i) va[i].info();

for(i=0; i<vh.size(); ++i) vh[i].info();

for(i=0; i<vc.size(); ++i) vc[i].info();

for(i=0; i<ve.size(); ++i) ve[i].info();

}

};

// Базовый класс ArmyBuilder объявляет интерфейс для поэтапного

// построения армии и предусматривает его реализацию по умолчанию

class ArmyBuilder

{

protected:

Army\* p;

public:

ArmyBuilder(): p(0) {}

virtual ~ArmyBuilder() {}

virtual void createArmy() {}

virtual void buildInfantryman() {}

virtual void buildArcher() {}

virtual void buildHorseman() {}

virtual void buildCatapult() {}

virtual void buildElephant() {}

virtual Army\* getArmy() { return p; }

};

// Римская армия имеет все типы боевых единиц кроме боевых слонов

class RomanArmyBuilder: public ArmyBuilder

{

public:

void createArmy() { p = new Army; }

void buildInfantryman() { p->vi.push\_back( Infantryman()); }

void buildArcher() { p->va.push\_back( Archer()); }

void buildHorseman() { p->vh.push\_back( Horseman()); }

void buildCatapult() { p->vc.push\_back( Catapult()); }

};

// Армия Карфагена имеет все типы боевых единиц кроме катапульт

class CarthaginianArmyBuilder: public ArmyBuilder

{

public:

void createArmy() { p = new Army; }

void buildInfantryman() { p->vi.push\_back( Infantryman()); }

void buildArcher() { p->va.push\_back( Archer()); }

void buildHorseman() { p->vh.push\_back( Horseman()); }

void buildElephant() { p->ve.push\_back( Elephant()); }

};

// Класс-распорядитель, поэтапно создающий армию той или иной стороны.

// Именно здесь определен алгоритм построения армии.

class Director

{

public:

Army\* createArmy( ArmyBuilder & builder )

{

builder.createArmy();

builder.buildInfantryman();

builder.buildArcher();

builder.buildHorseman();

builder.buildCatapult();

builder.buildElephant();

return( builder.getArmy());

}

};

int main()

{

Director dir;

RomanArmyBuilder ra\_builder;

CarthaginianArmyBuilder ca\_builder;

Army \* ra = dir.createArmy( ra\_builder);

Army \* ca = dir.createArmy( ca\_builder);

cout << "Roman army:" << endl;

ra->info();

cout << "\nCarthaginian army:" << endl;

ca->info();

// ...

return 0;

}



**Архитектура Dropbox**

Dropbox — [облачное хранилище данных](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BB%D0%B0%D1%87%D0%BD%D0%BE%D0%B5_%D1%85%D1%80%D0%B0%D0%BD%D0%B8%D0%BB%D0%B8%D1%89%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), принадлежащее компании Dropbox Inc., позволяющее пользователям хранить свои данные на серверах в *облаке* и делиться ими с другими пользователями в [Интернете](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82). Работа построена на [синхронизации данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)).

Dropbox позволяет пользователю размещать файлы на удалённых [серверах](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%B0%D0%BF%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)) при помощи [клиента](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) или с использованием [веб-интерфейса](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81)через [браузер](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80) (см. рис 3). При установке клиентского программного обеспечения Dropbox на компьютере создается синхронизируемая папка. Хотя главный акцент технологии делается на синхронизацию и обмен информацией, Dropbox ведёт историю загрузок, чтобы после удаления файлов с сервера была возможность восстановить данные. История изменения файлов ведётся по принципу [diff-кодирования](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D1%8C%D1%82%D0%B0-%D0%BA%D0%BE%D0%B4%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), чтобы сэкономить место, занимаемое [файлами](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0). В истории изменения записывается только отличие одной версии файла от другой.

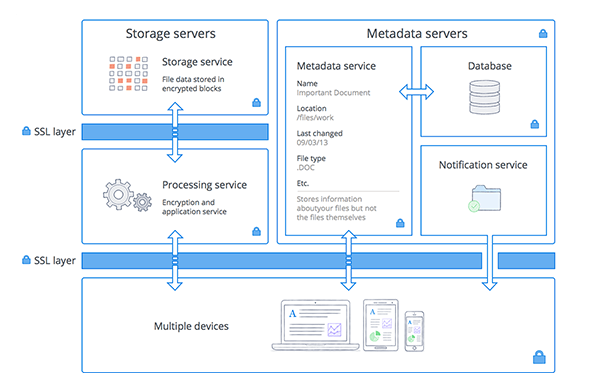


Рисунок 3

Существуют официальные клиенты Dropbox для таких платформ, как [Windows](https://ru.wikipedia.org/wiki/Windows), [Mac OS](https://ru.wikipedia.org/wiki/Mac_OS) и [Linux](https://ru.wikipedia.org/wiki/Linux), а также на мобильные: [iPhone](https://ru.wikipedia.org/wiki/IPhone), [iPad](https://ru.wikipedia.org/wiki/IPad), [Android](https://ru.wikipedia.org/wiki/Android), [BlackBerry](https://ru.wikipedia.org/wiki/BlackBerry), [Symbian](https://ru.wikipedia.org/wiki/Symbian_OS) и [Bada](https://ru.wikipedia.org/wiki/Bada). Ведётся разработка программы для [Windows Phone 7](https://ru.wikipedia.org/wiki/Windows_Phone_7).

Использованные технологии:

И Dropbox сервер и клиентское приложение написаны на Python. Клиент использует средства разработки GUI, такие как wxWidgets и Cocoa и другие важные библиотеки Python такие как Twisted, ctypes и pywin32. Dropbox зависит от librsync библиотеки, которая написана на C. Информация о файлах содержится в хранилище, реализованном на базе MySQL. Для хранения самих файлов используется Amazon S3.

Cтатистика:

Штат состоит из 110 сотрудников. Более 50 миллионов зарегистрированных пользователей. Каждые 3 минуты более чем миллион файлов сохраняются с использованием сервиса. 500 миллионов файлов сохраняются ежедневно.

Основной элемент системы — это блок (chunk) размером до 4 Mb. В случае, если файл большего размера, он разбивается на несколько блоков, и каждый блок воспринимается системой независимо от других. Для каждого блока вычисляется SHA256 хеш, и эта информация является частью метаинформации о файле. Dropbox уменьшает объем передаваемых данных за счет передачи только разницы между измененными блоками файла. Кроме того, локально он содержит всю метаинформацию по файлам, которую синхронизирует с сервером и передает только изменения с прошлой версии (incremental updates).   
Dropbox использует два типа серверов: управляющий (control) и сервер данных (data storage). Сервера управления находятся под контролем Dropbox, сервера данных — это сервера Амазона (Amazon S3, EC2). Для коммуникациями с серверами во всех случаях используется HTTPS.   
Доменные имена, используемые Dropbox, всегда заканчиваются на dropbox.com. В таблице 1 приведены поддомены для управляющих серверов и серверов данных. 

|  |  |  |
| --- | --- | --- |
| Поддомен | Хостинг | Описание |
| client-lb/clientX | Dropbox | Meta data |
| notifyX | Dropbox | Notifications |
| api | Dropbox | API control |
| www | Dropbox | Web servers |
| d | Dropbox | Event logs |
| dl | Amazon | Direct links |
| dl-clientX | Amazon | Client storage |
| dl-debugX | Amazon | Back traces |
| dl-web | Amazon | Web storage |
| api-content | Amazon | API storage |

Таблица 1

Dropbox: изнутри

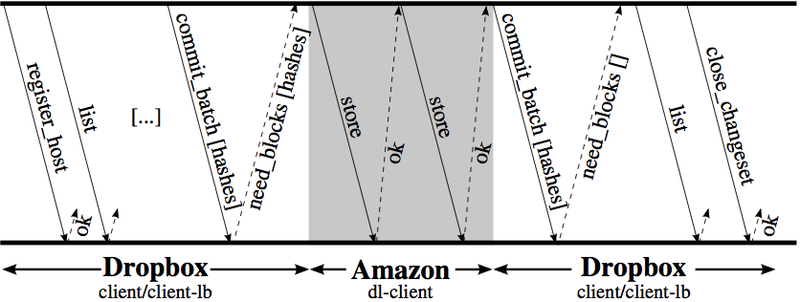
Поскольку Dropbox использует HTTPS для шифрования всего трафика между серверами, простой перехват не даст никакой полезной информации. Для исследования мы устанавливали Squid и направляли весь трафик с компьютера под Linux на этот прокси. Также на прокси поставили SSL-bump, чтобы можно было расшифровывать SSL. Последним шагом устанавливаем самоподписанный сертификат на Squid и изменяем сертификат внутри запущенного Dropbox приложения. Данная конфигурация позволяет расшифровать и просмотреть трафик Dropbox (см. рис 4).  


Рисунок 4

Иллюстрация показывает протокол, используемый Dropbox для загрузки локально измененных блоков на свои сервера. После регистрации клиента на управляющих серверах clientX.dropbox.com, команда list получает изменения в метаданных, которые показывают разницу между локальной копией и тем, что находится на сервере. Как только происходит локальное изменение файлов, Dropbox вызывает команду commit\_batch (client-lb.dropbox.com) и посылает измененные метаданные на сервер. После этого сервер отвечает, какие блоки ему необходимы, используя команду need\_blocks, и клиент отсылает эти блоки на Amazon (dl-clientX.dropbox.com). Сохранение каждого блока подтверждается командой ОК.  
После этого локальный клиент еще раз раз посылает команду commit\_batch на сервер и получает подтверждение, что все блоки получены. Транзакции сохранения данных могут выполняться параллельно.  
Протокол управления

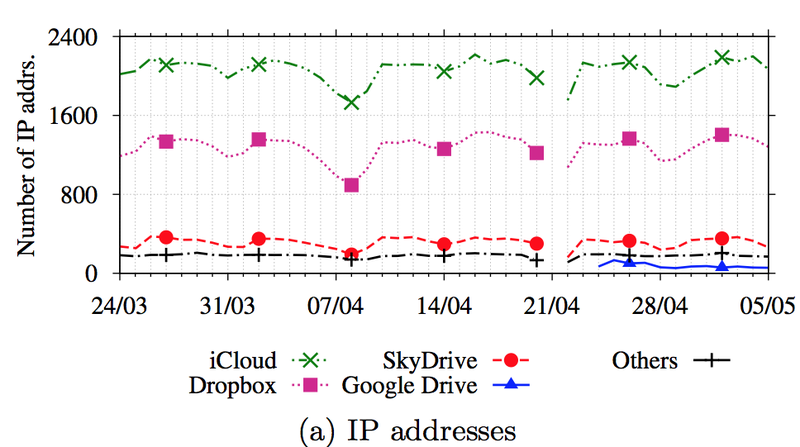
Dropbox использует следующие группы управляющих серверов:  
Уведомления (notifications).   
Dropbox держит постоянное открытое TCP соеденинение с серверами уведомлений (notifyX.dropbox.com). Это необходимо для получения информации об изменении файлов, которое могло произойти на других клиентах. По сравнению с другим трафиком, эта информация не шифруется. Используется задержка HTTP ответа для быстрого уведомления клиентов (push mechanism). Клиент посылает запрос, и сервер задерживает ответ примерно на 60 секунд. По истечении 60 секунд, клиент немедленно посылает следующий запрос на сервер. Если ответ сформирован раньше, то сервер отвечает немедленно.

* Управление метаданными (meta-data administration)  
  Сервера управления метаданными отвечают не только за информирование об изменениях в блоках и файлах, но также и за авторизацию (authentication) клиента. Для этих серверов используются следующие доменные имена: client-lb.dropbox.com, clientX.dropbox.com. Кроме этого, сервера управления могут контролировать поведение клиента. В момент эксперимента было замечено, что сервера могут указать клиенту максимальное количество блоков, которое он может посылать на сервер. Это используется для управления трафиком, который генерирует клиент.
* Системные сообщения (system logs)   
  сервера предоставляются Амазоном и имеют название dl-debug.dropbox.com; остальные сообщения идут непосредственно на Dropbox d.dropbox.com.

Набор данных и популярность клиентов(см. табл. 2).

|  |  |  |  |
| --- | --- | --- | --- |
| Для сбора трафика использовался open source инструмент Tstat. Tstat позволяет собирать разнообразную информацию о ТСР, предоставляя сведения более, чем о сотне разнообразных параметров соединения. Поскольку Dropbox использует HTTPS, было установлено, что имя во всех сертификатах, используемых Dropbox — \*.dropbox.com. Это было важно для правильной классификации трафика. Была пополнена открытая информация записями с серверов DNS, к которым обращались клиенты. Таким образом были связаны IP адреса и имена серверов. Tstat возвращал незашифрованную информацию об устройстве и именах директорий, которыми обменивался клиент и сервер уведомлений.  Данные были получены с помощью установки Tstat в 4 точках в Европе. Записи с точек, обозначенных как Home 1 и Home 2, составляют данные пользователей известного интернет-провайдера (ISP), предоставляющего интернет по ADSL и оптическому кабелю. Данные, обозначенные как Campus 1 и Campus 2, были собраны в университетах. Исследования проводились с 24 Марта 2012 по 5 Мая 2012.  Имя | Тип | Количество IP адресов | Обьем данных (GB) |
| Campus 1 | Wired | 400 | 5,320 |
| Campus 2 | Wired/Wireless | 2,528 | 55,054 |
| Home 1 | FTTH/ADSL | 18,785 | 509,909 |
| Home 2 | ADSL | 13,723 | 301,448 |

Таблица 2

Ниже приведен график, который показывает, сколько различных IP адресов связывалось с облачным сервисом хранения хотя бы раз в день(см. рис 5).  
  
Рисунок 5

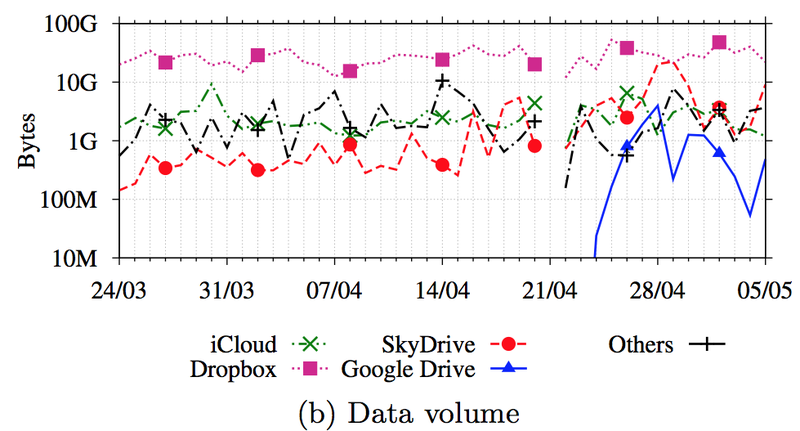
Второй график показывает, сколько данных было передано на это облачное хранилище в день (см. рис 6).  
  


Рисунок 6

Хотелось бы обратить внимание на следующее:  
Несмотря на большое количество устройств, использующих iCloud, количество данных, передаваемое на этот сервис, соизмеримо с другими сервисами.

* В момент появления Google Drive, трафик, передаваемый на этот сервис, сделал большой скачок и приблизился к iCloud; в то же время количество инсталляций программы оставалось минимальным.

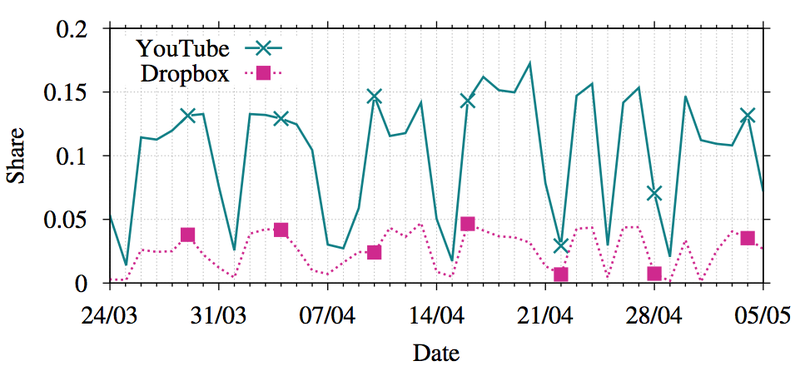
Для сравнения приведем данные использования сервисов YouTube и Dropbox в Campus 2 (см рис. 7).  
  


Рисунок 7

Таблица показывает суммарный трафик Dropbox, который мы отследили в ходе наших измерений (см. табл 3).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Campus 1 | Campus 2 | Home 1 | Home 2 | Всего |
| Запросов | 167,189 | 1,902,824 | 1,438,369 | 693,086 | 4,204,666 |
| Обьем (GB) | 146 | 1,814 | 1,153 | 506 | 3,624 |
| Устройств | 283 | 6,609 | 3,350 | 1,313 | 11,561 |

Таблица 3

Анализ трафика

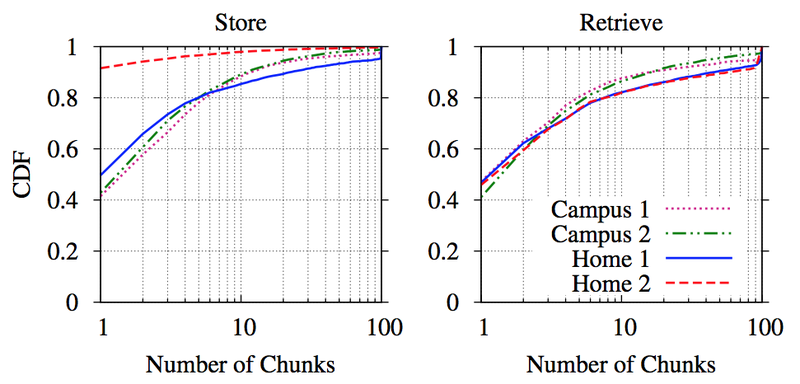
Графики показывают куммулятивную функцию распределения для различного количества блоков(см. рис 8).   
  


Рисунок 8

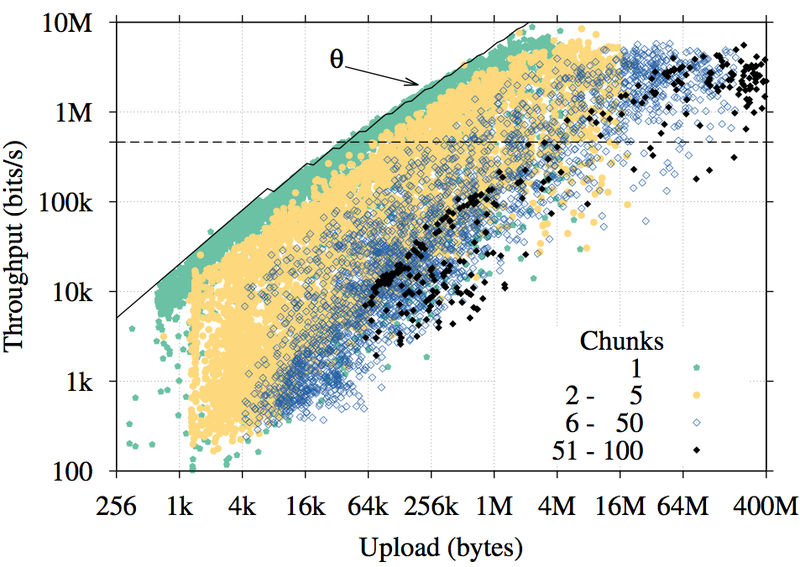
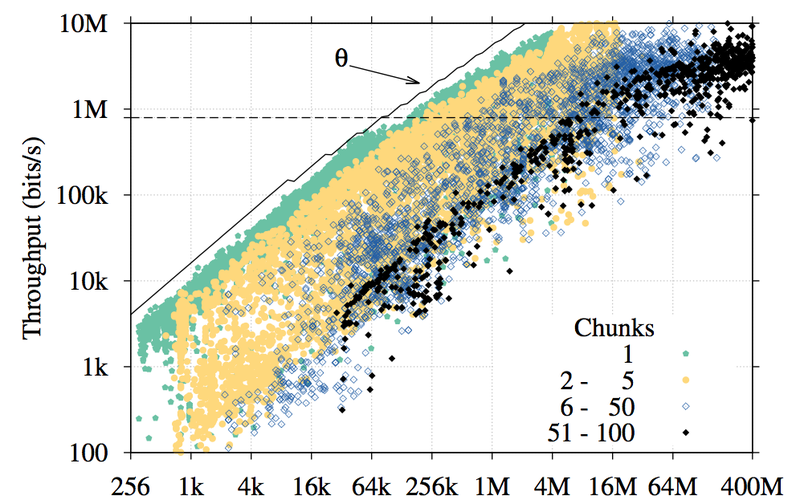
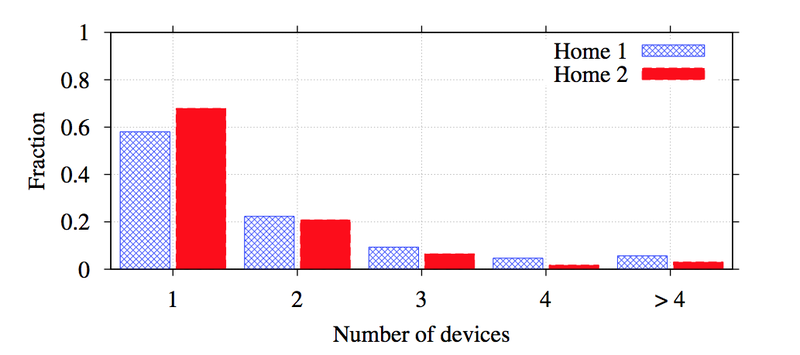
Оказалось, что более, чем в 80% процентах случаев, количество блоков при сохранении данных не превышает 10. График для данных с точки Home 2 существенно отличается от остальных, так как здесь мы наблюдали одного клиента, который постоянно, на протяжении нескольких дней, пересылал одни и те же блоки. Анализ полученных данных показывает, что основной сценарий использования Dropbox — это постоянная работа с небольшими, постоянно изменяемыми файлами.  
Как мы рассмотрели выше, Dropbox использует центральные сервера для хранения данных. Это сразу наводит на вопрос о скорости работы сервиса для пользователей, которые находятся географически далеко от серверов.  
Максимальная скорость, которая наблюдаласт, была близка к 10 Mbit/s и наблюдалась на файлах с размером больше 1 Mb. Средняя скорость для Campus 2 была: запись — 462 kbits/s и чтение — 797 kbits/s. Для Campus 1: запись — 359 kbits/s и чтение — 783 kbits/s (см. рис 9, 10).   
  


Рисунок 9  
  
Рисунок 10

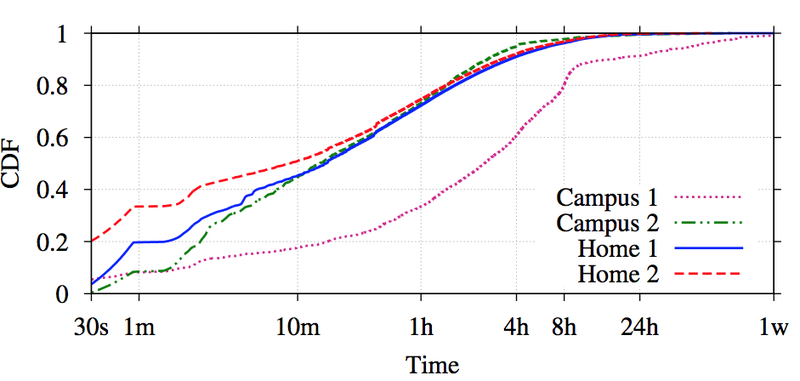
Также из графиков видно, что скорость существенно зависит от количества блоков: чем больше блоков, тем ниже скорость.  
Изменения в Dropbox 1.4.0

Начиная с версии 1.4.0, Dropbox добавил две новые команды: store\_batch иretrieve\_batch, что позволяет работать с несколькими блоками одновременно. Это улучшение должно существенно улучшить пропускную способность сервиса.  
Количество устройств

График показывает количество установок Dropbox у пользователей дома. Примерно в 60% случаев у пользователей существует только 1 устройство с Dropbox. У 25% пользователей дома есть 2 устройства, использующих Dropbox(см.рис 11).  
  
Рисунок 11

Среднее время использования

График показывает среднее время использования Dropbox. Анализируя время использования, мы смотрели, сколько времени клиент поддерживал связь с сервером уведомлений. Поскольку клиент всегда держит это соединение открытым либо открывает его заново, это хороший способ оценить время использования (см. рис 12).

  
Рисунок 12

Из графика видно, что время использования Dropbox в большинстве случаев меньше 4 часов. Исключение составляет Campus 1, где много рабочих компьютеров и компьютеров, работающих постоянно.  
Исходные данные

Вы можете загрузить исходные данные, которые использовались в этой статье для дальнейшего анализа. ([Исходные данные](http://traces.simpleweb.org/dropbox/)).  
Хочу обратить внимание, что оригинальная статья содержит больше информации. В ней могут быть ответы на вопросы, которые могут возникнуть у вас после прочтения.

Список литературы и источников:

1. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес Приемы объектно-ориентированного проектирования. Паттерны проектирования = Design Patterns: Elements of Reusable Object-Oriented Software. — СПб: [«Питер»](http://ru.wikipedia.org/wiki/%D0%9F%D0%B8%D1%82%D0%B5%D1%80_%28%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE%29), 2007. — С. 366.
2. Алан Шаллоуей, Джеймс Р. Тротт Шаблоны проектирования. Новый подход к объектно-ориентированному анализу и проектированию = Design Patterns Explained: A New Perspective on Object-Oriented Design. — М.: [«Вильямс»](http://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%BB%D1%8C%D1%8F%D0%BC%D1%81_%28%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE%29), 2002. — С. 288.
3. Разработка высоконагруженного веб-приложения. Плотников Д.Ю., Малёваный Е.Ф., Аноприенко А.Я.
4. Информационные управляющие системы и компьютерный мониторинг (ИУС и КМ 2012)— Донецк, ДонНТУ — 2011, с. 431-435
5. [www.dropbox.com](http://www.dropbox.com)
6. <http://www.simpleweb.org/wiki/Dropbox_Traces>
7. ru.wikipedia.org