УДК 004.4

В. Ю. Коцюбинский, к. т. н., доц.; Р. С. Головаха

УСОВЕРШЕНСТВОВАНИЕ ПРОЦЕССА ПЕРЕДАЧИ СООБЩЕНИЙ С ИСПОЛЬЗОВАНИЕМ FIX/FAST ПРОТОКОЛА В ФИНАНСОВЫХ КЛИЕНТ-СЕРВЕРНЫХ СИСТЕМАХ

Рассмотрена проблема возникновения задержек при передаче больших объемов информации в сфере электронной торговли на финансовых рынках, исследованы современные методы организации финансовых данных, реализован и осуществлен детальный анализ подходов решения описанной проблемы.

Ключевые слова: *передача данных, FIX-протокол, FAST-протокол.*

Введение. С бурным развитием индустрии информационных технологий и распространения сети Интернет большую популярность приобрела сфера электронной торговли на финансовых рынках. Благодаря современным средствам автоматизации данного процесса трейдеру больше нет необходимости находиться непосредственно на бирже, так как сейчас он может покупать тот или иной актив со своего торгового терминала, отдавая приказы через Интернет.

Широкое распространение электронной торговли на финансовых рынках обеспечило существенное развитие технологий в данной сфере.

Анализ предыдущих исследований. В 1992 году был разработан Financial Information eXchange (FIX) протокол, для организации структуры финансовых данных, которые передаются. FIX-протокол является международным стандартом для обмена финансовой информацией между участниками биржевых торгов в режиме реального времени. FIX-протокол поддерживается большинством крупнейших банков и бирж мира, а также электронными системами для интернет-трейдинга [1].

Использование FIX определяет ряд преимуществ, таких как:

– идеальная структура передаваемых данных делает FIX-сообщение легкими для чтения как программным кодом, так и человеком. Сообщения FIX состоят из набора полей, которые отделяются друг от друга специальным кодом;

– FIX-протокол позволяет осуществить проверку целостности передаваемых данных, поскольку определяет длину тела сообщения и контрольную сумму всех байтов;

– поддержка возвращения потерянных данных. FIX-протокол определяет поле, которое означает порядковый номер сообщения. Если последовательность данных нарушена, любая из сторон может выслать запрос о потерянных сообщениях;

– безопасность обмена данных. FIX-протокол поддерживает авторизацию обеих сторон и шифрования данных по разным методам [2].

С течением времени объемы торговли финансовыми активами значительно возросли, что привело к увеличению объема передаваемых данных, возникновению значительных задержек при передаче сообщений и несостоятельности клиент-серверных систем работать в реальном времени.

В 2004 году в Нью-Йорке на конференции компании FPL (FIX Protocol, Ltd. обладает правами и поддерживает спецификацию FIX-протокола) обсуждалась проблема о возникновении задержек при передаче сообщений, вызванная увеличением объемов передаваемых данных в финансовых сетях. Формат классических полей FIX-протокола считается достаточно объемным, и их обработка является накладной. Задержка в доставке данных привела к нарушению возможности ведения торгов в реальном времени, что в свою очередь привело к неспособности трейдеров торговать вообще. Таким образом,

FIX-протокол подлежал оптимизации.

В 2006 году была выпущена первая версия FAST-протокола (FAST 1.0). FAST (FIX Adapted for STreaming) протокол – это технология, направленная на оптимизацию представления данных в сети. Он используется для обеспечения высокой пропускной способности и малой задержки при передаче данных между финансовыми системами путем сжатия сообщений различными средствами [3 – 4].

FAST-протокол использует такие технологии сжатия данных:

– кодирование данных на уровне сообщений, а именно использование шаблонов. Шаблоны определяют передаваемые данные и их последовательность, что позволяет исключить передачу тегов, идентифицирующих поля;

– кодирование на уровне полей FIX-сообщения. FAST-протокол разделяет поля на различные типы, что позволяет или вообще избежать, или обеспечить частичную передачу соответствующих значений;

– ограничение объема по типу передаваемых данных;

– использование Stop Bit. Кодирование символа SOH, разделяющего поля FIX-сообщения, в значение самих закодированных данных [5].

FAST-протокол позволяет обеспечить значительное сжатие данных при их передаче, но возникает другая проблема – это дополнительные процессы при транспортировке сообщения, а именно кодирование и декодирование. При неэффективной реализации функциональности данных процессов, время на их выполнение будет достаточно большим, поэтому применение FAST может вообще потерять всякий смысл.

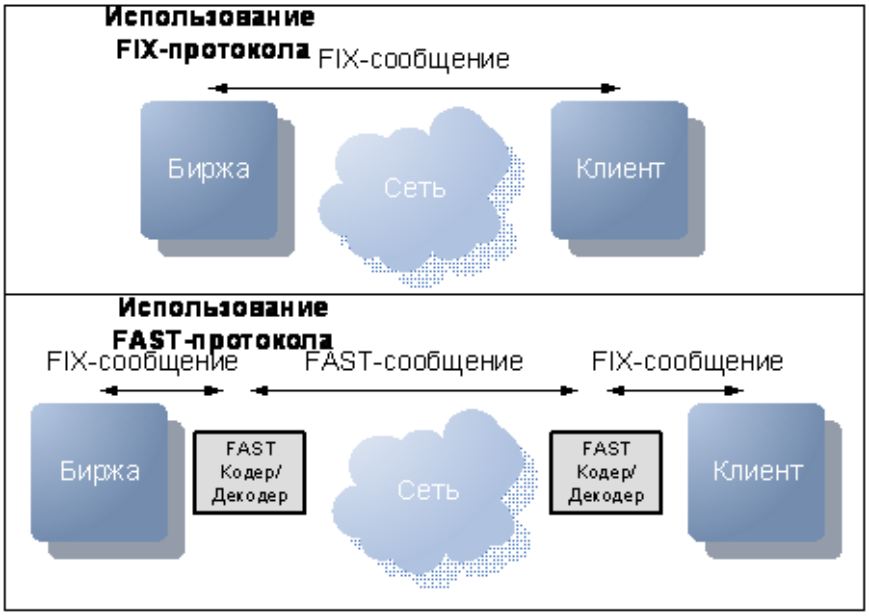

Рис. 1. Сравнение использования протоколов FIX и FAST при транспортировке сообщений

Современные электронные системы для интернет-трейдинга обеспечивают создание торговых операций через графический интерфейс, поэтому все финансовые данные представляются трейдеру в чистом виде. На программном уровне это означает, что после процесса декодирования система обрабатывает FIX-сообщения и передает соответствующие значения на отражение их пользователю.

Таким образом, полная система, которая обеспечивает передачу данных от биржевого сервера к их графическому представлению клиенту, требует наличия нескольких процессов, включающих в себя: чтение сообщений из информационного биржевого канала, декодирование данных, обработку сообщений, передачу соответствующих значений клиенту.

Так как электронная торговля на финансовых рынках происходит в реальном времени,

задержка при передаче данных должна быть достаточно малой, таким образом, актуальность минимизации времени на выполнение описанных выше процессов очевидна.

Целью работы является повышение пропускной способности системы при передаче финансовых данных от биржевого сервера до получения соответствующих значений клиентом.

Материалы и результаты исследований. Сейчас существуют известные способы оптимизации времени на выполнение рассмотренных выше процессов. Проведем детальный анализ и реализуем программно каждый из методов.

1. Распараллеливание потоков. То есть реализация программного кода с возможностью выполнения различных задач в различных потоках. Данный подход обеспечивает асинхронную работу нескольких процессов одновременно, что в свою очередь может повысить скорость выполнения кода в число раз, зависящее от количества созданных потоков.

С целью увеличения быстродействия выполнения задач целесообразно реализовывать программный код таким образом, чтобы одновременно работало не больше потоков, чем число ядер процессоров в системе. В противном случае задачи будут выполняться по очереди, что соответствует синхронному выполнению, поэтому потеряется всякий смысл использования параллельных потоков.

Так для разрабатываемой системы, целесообразно использовать четыре потока, которые будут осуществлять такую функциональность как:

– вычитка закодированных данных (FAST-сообщения) по информационному каналу;
– декодирование сообщений в формат FIX;
– обработка FIX-сообщений;
– передача соответствующих данных одному или нескольким клиентам.

Очевидно, что для рассматриваемой системы использование подхода распараллеливания потоков может повысить скорость программы, то есть уменьшить задержку при передаче данных от биржевого сервера до 4-ох раз (для 4-ох ядерного процессора).

2. Фабрика объектов. Для программ, главным предназначением которых является транспортировка данных, создание новых объектов является критическим моментом, так как способствует выделению памяти, что в свою очередь является трудоемким процессом, который замедляет всю систему. Поэтому во избежание лишних задержек целесообразно создавать все необходимые объекты при инициализации системы, что на практике не всегда возможно.

Так для рассматриваемой системы каждый раз при поступлении новых данных от биржевого сервера необходимо создавать новый объект FIX-сообщения. Данный объект будет хранить декодированные данные FAST-сообщения для их дальнейшей обработки. Фабрика объектов позволяет частично избавиться периодичности создания новых объектов. Реализация данной функциональности выполняется следующим образом:

– при инициализации системы создается определенное количество FIX-объектов, что будут использованы по мере необходимости;
– после использования всех объектов, созданных во время инициализации, при последующих запросах к фабрике каждый раз будут создаваться новые объекты.

Очевидно, что в любом случае данный подход имеет преимущества, то есть влияет на уменьшение времени при транспортировке данных к клиенту. Но данная величина прямо пропорциональна количеству передаваемых сообщений от биржевого сервера и размера, соответственно, самой фабрики объектов, т. е. количества объектов, которые будут созданы при инициализации системы.

Проведем тестирование фабрики, извлекая 1 млн. новых FIX-объектов:

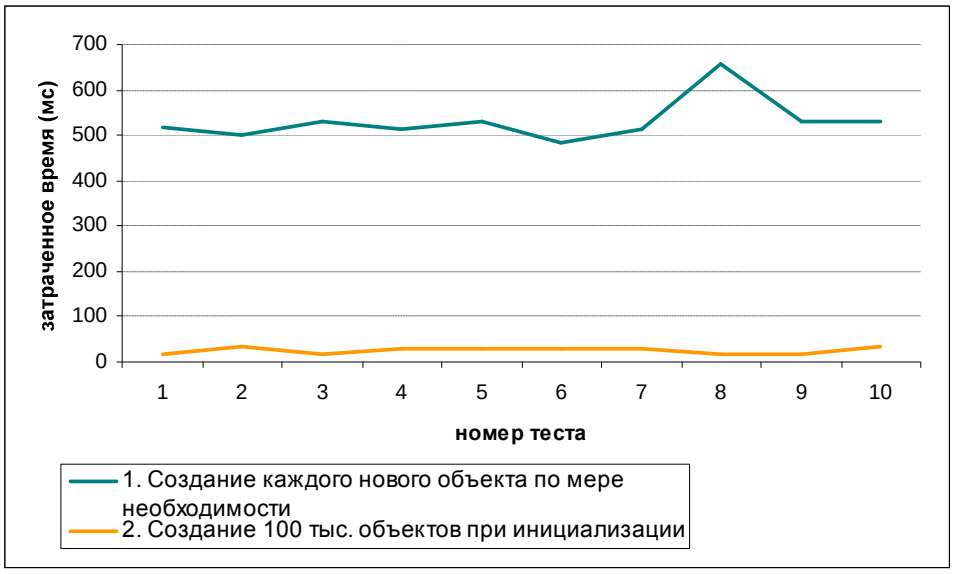

Рис. 2. Анализ применения фабрики объектов

Результаты теста на рис. 2 показывают, что время, потраченное на создание нового объекта каждый раз по необходимости, гораздо больше (примерно в 20 раз) по сравнению со временем, которое приближается к нулю при использовании фабрики объектов.

Мы можем определить любой размер фабрики, то есть количество объектов, которые будут создаваться при инициализации системы, что будет влиять лишь на оперативную память.

3. Работа непосредственно с массивами байтов и применение побитовых операций. Поскольку вся цифровая информация в компьютерных системах представляется в виде байтов, очевидно, что работа непосредственно с ними требует наименьших затрат времени по сравнению с приведением байтов до тех или иных типов данных, которые являются более удобными для работы.

Так как перед нами стоит задача повышения пропускной способности системы передачи данных, т. е. уменьшение времени на обработку сообщений, целесообразно, по возможности, обеспечить работу программного кода непосредственно с байтами. Применим данный подход при декодировании входных данных, т. е. FAST-сообщений, поступающих из биржевого сервера в виде массивов байтов:

1)
```
offset=0;
while (offset < buffer.length-1) {
   value = (value << 7) | (buffer[offset++]);
}
value = (value << 7) | (buffer[offset++] & Byte.MAX_VALUE);
```

2)
```
Integer intDecodedVal = Integer.parseInt(strBuffer);
StringBuilder intDecBinSb = new StringBuilder(Integer.toBinaryString(intDecodedVal));
for(int i = intDecBinSb.length()-8; i >=0; i-=8) {
   intDecBinSb.deleteCharAt(i);
}
int value = Integer.parseInt(intDecBinSb.toString(), 2);
```

Работа непосредственно с массивами байтов в примере 1) позволяет применить побитовые операции, которые вместе дают значительные преимущества при использовании (примерно в

10 раз по сравнению с примером 2)). Поскольку очевидно, что программный код примера 2) включает в себя лишние операции, такие как приведение байтов в двоичный код, что является лишним по сравнению с примером 1) и создание новых объектов, что было описана в пункте 2.

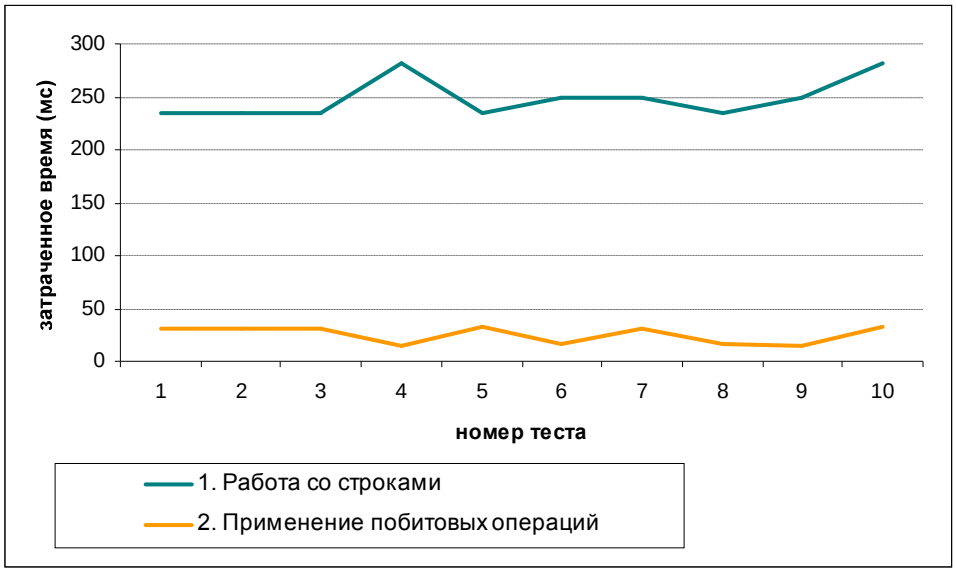

Рис. 3. Анализ применения побитовых операций

4. Лишение универсальности программного кода. Очевидно, что универсальность кода в значительной степени уменьшает его объем, но в то же время и уменьшает его быстродействие. Для большинства программ, данный момент не имеет критического значения, поэтому ему не уделяется особое внимание. Но для программных систем транспортировки сообщений, где поток данных является высоким и главный критерий оценки – скорость передачи, универсальность кода может значительно повлиять на время обработки информации.

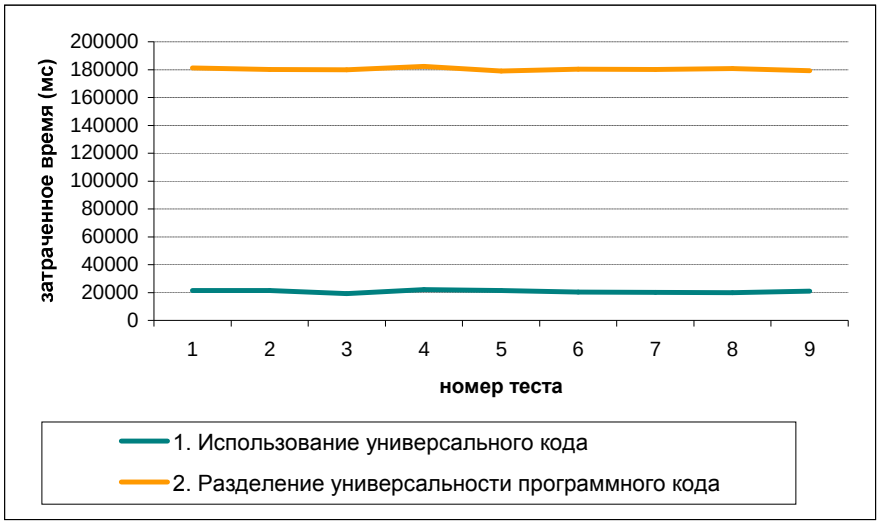

Рис. 4. Анализ применения универсальности кода

В разрабатываемой системе для достижения наиболее эффективного результата необходимо обеспечить разделение всей универсальной функциональности. Реализуем данный подход для процесса декодирования сообщений, так как FAST-протокол обеспечивает распределение всех данных на категории, которое определяются следующим

образом:

1) Fields operators. FAST-протокол описывает 7 типов операторов полей, что определяют последовательность действий, которые должны применяться к тому или иному полю FIX-сообщения;
2) Nullable fields. Для таких полей декодированное значение "0" для числовых типов данных или "" для строчных означает передачу значения "null", которое имеет различное применение в соответствии к другим параметрам поля;
3) Data types. FAST протокол описывает 6 типов данных, что требуют соответствующие правила, которые применяются для их декодирования.

Таким образом, разделив данную функциональность, получим $7 \cdot 2 \cdot 6 = 84$ класса, которые определяют достаточно большую объемность, но в то же время значительный выигрыш во времени (примерно в 20 раз, что представлено на рис. 4).

Данные исследования отображают результаты усовершенствования существующих подходов к реализации процесса FAST-кодирования / декодирования по заказу компании jettekfix.com.

Выводы. В данной работе рассмотрены проблемы, которые являются актуальными при передаче больших объемов данных в системах электронной торговли на финансовых рынках, оптимизированы современные подходы решения данных проблем, и проведен анализ их использования. В результате выполнения данного исследования доказано, что использование представленных приемов позволяет повысить пропускную способность системы передачи данных от биржевого сервера примерно в 4 раза, и, в частности, процесса декодирования FAST-сообщений в 50 раз.

СПИСОК ЛИТЕРАТУРЫ

1. FIX protocol [Електронний ресурс] // Режим доступу: http://ru.wikipedia.org/wiki/Financial_Information_eXchange.

2. Lamoureux Robert. Financial information exchange protocol / Robert Lamoureux // FIX Protocol Ltd. – 2001. – 280 p.

3. FAST protocol [Електронний ресурс] // Режим доступу: http://en.wikipedia.org/wiki/FAST_protocol.

4. Rosenborg David. FAST Protocol Technical Overview. / David Rosenborg. London: FIX Protocol Ltd. – 2006. – 10 p.

5. Rosenborg David. FAST Specification / David Rosenborg. London: FIX Protocol Ltd. – 2006. – 45 p.

Коцюбинский Владимир Юрьевич – к. т. н., доцент кафедры автоматики и информационно-измерительной техники.

Головаха Роман Сергеевич – студент кафедры автоматики и информационно-измерительной техники.

Винницкий национальный технический университет.