УДК 004.738
ББК 32.81

ИНТЕРНЕТ-СЛУЖБА ЗАЩИЩЕННОЙ ОБРАБОТКИ ИНФОРМАЦИОННЫХ ЗАПРОСОВ

Асратян Р. Э.[1]
(*Учреждение Российской академии наук
Институт проблем управления РАН, Москва*)

Рассмотрены принципы организации новой сетевой службы, предназначенной для реализации защищенной обработки информационных запросов в распределенных информационно-управляющих системах. Отличительными особенностями службы являются высокая скорость обработки, тесная интеграция функций аутентификации и защиты данных с функциями сетевого информационного взаимодействия, а также возможность отладки как клиентских, так и серверных компонент системы вне сетевой среды.

Ключевые слова: распределенные системы, Интернет-технологии, Web-технологии, информационное взаимодействие, информационная безопасность.

1. Введение

Уже более десяти лет сетевая архитектура .NET на базе Web-сервисов, опирающихся на WS и WCF технологии, занимает почти монопольное положение в разработках распределенных информационно-управляющих систем. Решающую роль здесь играет мощная и многоязыковая поддержка этой архитектуры в целом ряде современных систем программирования (MS Visual Studio, Borland C++ builder, Oracle developer suit и т.п.) и Web-серверов (IIS, Apache, Oracle HTTP server). Этому же спо-

[1] *Рубен Эзрасович Асратян, кандидат технических наук, доцент* (rea@ipu.*ru*).

собствуют и удачные сетевые стандарты WSDL и SOAP, сопровождающие жизненный цикл сетевых приложений от разработки до эксплуатации и обеспечивающие большую гибкость в спецификациях методов (функций-членов) электронных сервисов [3,6].

Однако в последние годы появилась тенденция поиска альтернатив технологии Web-сервисов в разработках информационно-управляющих систем. В качестве основных причин можно назвать:

- возрождение интереса к традиционной для классического HTTP модели взаимодействия на основе обмена сообщениями (вместо характерной для архитектуры .NET модели, основанной на вызове методов удаленных объектов),
- отсутствие в архитектуре .NET встроенных средств защиты и аутентификации сетевых сообщений,
- отсутствие средств отладки клиентских и сервисных компонент системы (и их взаимодействия) вне сетевой среды и Web-сервера.

Рассмотрим ситуацию, в которой клиент посылает сложный (включающий десятки параметров) информационный запрос электронному сервису и получает в результате текстовый документ отчета, к которому могут быть приложены один или несколько графических документов. С ростом сложности информационных запросов разработчики информационно-управляющих систем все чаще выбирают формат XML-документа как для представления параметров запроса, так и для представления результатов его выполнения. Модель обмена текстовыми и двоичными сообщениями представляется в данном случае более простой и адекватной, чем модель вызова удаленных методов через SOAP. Во всяком случае, она позволяет избежать явной избыточности, связанной с упаковкой документов в объемлющую SOAP-структуру (например, к перекодировке всех символов «<» в последовательности «<» и т.п. в случае передачи документа в форме символьной строки, а при наличии двоичных приложений к документу – к обязательной текстовой кодировке base64). Вместе с тем, появились признаки

того, что компактность представления данных в сети вновь стала заботить разработчиков (например, в относительно новом сетевом протоколе WebSocket [8] сделан акцент на экономии сетевого трафика, в том числе за счет возможности передачи данных в двоичной форме).

Основной метод организации защиты и аутентификации данных в .NET, не требующий дополнительного программирования, основан на использовании протокола HTTPS в качестве «транспорта» для передачи SOAP-сообщений через сеть. Однако, этот метод скорее основан на создании защищенных каналов взаимодействия (с помощью технологии SSL/TLS [5]), чем не на защите отдельных сообщений. Поэтому, его использование сопряжено с рядом важных ограничений: невозможность анализа и регистрации сообщений (информационных запросов и ответов) вместе с удостоверяющими их электронными цифровыми подписями (ЭЦП), недопустимость использования нескольких ЭЦП (например, исполнителя и руководителя) для одного сообщения и т.п. [1]. Это существенно усложняет задачу разработчика, так как внимание к проблемам информационной безопасности непрерывно возрастает [2,7].

В данной статье рассматриваются принципы организации новой сетевой службы PMS (Protected Message Service), в разработке которой сделана попытка преодоления вышеперечисленных недостатков. Суть подхода заключается в тесной интеграции функций сетевого информационного обмена с функциями защиты и аутентификации данных. Чисто внешне эта интеграция проявляется в том, что эти функции реализованы в наборе методов одного программного класса – главного класса PMS «Защищенное сообщение», отображающего электронный документ (информационный запрос или ответ), снабженный одной или несколькими удостоверяющими ЭЦП. В отличие от технологии Web-сервисов описываемая служба опирается не на модель вызова методов удаленных объектов, а на модель обмена сообщениями. В данном случае это означает, что все сервисные обрабатывающие функции (методы) имеют одинаковую, жесткую спецификацию: они получают объект класса «Защищенное сообщение» в качестве параметра и возвращают объект того же

класса. Эти обрабатывающие функции группируются в одну или несколько динамических библиотек, которые подключаются к серверу PMS в момент его запуска (каждая библиотека может рассматриваться как отдаленный аналог Web-сервиса в .NET) и становятся доступными для клиентских компонент. Web-технология (HTTP, HTTPS, SOAP, WSDL и т.п.), как и Web-серверы не используются вовсе.

2. Два коротких примера

Как и всякая сетевая служба, основанная на базовом сетевом протоколе TCP/IP [4], PMS поддержана клиентским и серверным программным обеспечением. Сервер PMS представляет собой постоянно активную программу, обслуживающую запросы на обработку от клиентов (по умолчанию используется порт 8132). Клиентское программное обеспечение представляет собой библиотеку функций PmsBase.dll, реализующих прикладной программный интерфейс (API) к PMS. Этот интерфейс является «лицом» PMS с точки зрения пользователя. Чтобы сразу же дать представление о нем, рассмотрим два коротких фрагмента кода в нотации языка C#.

На рис. 1 проиллюстрировано простое обращение к обрабатывающей функции без применения средств защиты данных. Две первые строки кода определяют две переменные типа PmsMessage, представляющего собой главный класс PMS («Защищенное сообщение»). Первой переменной (Request) присваивается значение: объект, инициированный символьной строкой (например, содержащей XML-документ информационного запроса); вторая переменная (Reply) предназначена для результата обращения. В третьей строке определяется и инициируется переменная класса PmsConnection, предназначенного для создания и прекращения сетевого соединения с сервером.

Собственно, вызов обрабатывающей функции выполняется в четвертой и пятой строках. Сначала устанавливается соединение с сервером с помощью метода Connect (с указанием сетевого имени или адреса), а затем выполняется вызов удаленной функции с помощью применения метода Process к переменной

Request на основе данного соединения. Предполагается, что к серверу PMS подключена библиотека MyLib.dll, содержащая код функции MyFunc.

В остальных строках кода осуществляется запись результата обращения в стандартный вывод (предполагается, что он, как и запрос, имеет форму символьной строки) и закрытие соединения с сервером с помощью метода Disconnect, так как в данном примере оно больше не нужно.

```
PmsMessage Request=  new PmsMessage("<doc> … </doc>");
PmsMessage Reply;
PmsConnection MyConn = new PmsConnection();
MyConn.Connect("MyServer");
Reply=Request.Process(MyConn, "MyLib.MyFunc");
if(Reply != null)
    Console.WriteLine (Reply.GetString());
else
    Console.WriteLine ("Ошибка: " + MyConn.ErrMsg);
MyConn.Disconnect();
```

Рис. 1. Пример использования PMS без защиты данных

Рассмотрим теперь другой пример кода, который выполняет обращение к той же самой сервисной функции, но с использованием средств защиты (см. рис. 2). Легко видеть, что второй пример кода получен путем добавления к первому нескольких добавочных строк, которые выделены серым фоном. Первые две из добавленных строк определяют две новые переменные класса PmsCertList, предназначенного для хранения в памяти списков сертификатов с открытыми ключами в стандарте X509 и содержащего несколько конструкторов для загрузки сертификатов из файлов или из системных хранилищ с поиском по имени владельца или серийному номеру. В переменную SendCert загружаются сертификаты, соответствующие именам владельцев «Иванов» и «Петров» для последующего формирования двух ЭЦП в запросе. Это означает, что с этими сертификатами (вернее, с содержащимися в них открытыми ключами)

обязательно должны быть связаны парные им закрытые ключи, иначе формирование ЭЦП закончится неудачно. В переменную RecvCert загружается один сертификат для выполнения шифрования.

Формирование ЭЦП выполняется применением метода AddSignations к переменной Request, а шифрование данных перед передачей в сеть – всего лишь добавлением дополнительного параметра в вызов метода Process. Последующие добавленные строки обеспечивают последовательную проверку всех ЭЦП в полученном ответе сервера и запись в стандартный вывод сведений о подписантах.

```
PmsMessage Request=  new PmsMessage(“<doc> … </doc>“);
PmsMessage Reply;
PmsConnection MyConn = new PmsConnection();
PmsCertList  SendCert = PmsCertList( new string []{“Иванов“,
    “Петров“});
PmsCertList  RecvCert = PmsCertList( new string [] {“Admin“});
Request.AddSignations(SendCert);
MyConn.Connect(“MyServer“);
Reply=Request.Process(MyConn, “MyLib.MyFunc“, RecvCert);
if(Reply != null)
{
    string Signer;
    for(int i=0; Reply.Signatures.Length; i++)
        if(Reply.VerifySignation(i, out Signer) >= 0)
            Console.WriteLine “Ответ подписал: “ + Signer);
    Console.WriteLine (Reply.GetString());
}
else
    Console.WriteLine (“Ошибка: “ + MyConn.ErrMsg);
MyConn.Disconnect();
```

Рис. 2. Пример использования PMS с защитой данных

Разумеется, из приведенных примеров намеренно удалены операторы обработки ошибок и исключительных ситуаций.

3. Принципы организации PMS

Как видно из приведенных примеров, вся функциональность PMS на стороне клиента основана на трех главных классах: PmsMessage (защищенное сообщение), PmsConnection (соединение с сервером PMS) и PmsCertList (список сертификатов). Общая особенность всех трех классов заключается в том, что они содержат очень мало элементов данных (полей) и сравнительно много функций-членов (методов). В таблице 1 приведены основные поля данных в нотации C#.

Таблица 1. Основные программные классы PMS

Название класса и основные поля данных	Описание
Класс: PmsMessage Основные поля: public byte [] Data; public Signature [] Signatures;	Член Data представляет собой «контейнер» данных в форме массива байтов. В классе имеется множество методов для загрузки в член Data (и обратно) символьных строк, массивов строк, наборов файлов (вместе с их названиями) и т.п. Член Signatures представляет собой массив электронных подписей - объектов класса Signature. Каждый такой объект содержит свертку (digest) массива Data, зашифрованную закрытым ключом подписанта, и сертификат структуры X509, содержащий открытый ключ. Важно отметить, что пользователю совершенно необязательно знать устройство этого класса. Методы класса данных позволяют добавлять новые подписи, проверять и удалять их, а также отправлять данные на обработку в сервер PMS с

	возможностью автоматической шифрации запроса и дешифрации результата обработки.
Класс: PmsConnection Основные поля: Socket msock; string Host; int Port; public string ErrStr;	Основным членом класса является msock, в котором хранится дескриптор сетевого соединения и ряд настроек типа таймаутов чтения из сети и записи в сеть. Члены Host и Port содержат адресные данные последнего соединения (для автоматического восстановления его в случае разрыва). Член ErrStr содержит последнее диагностическое сообщение, возникшее при неудачной попытке соединения с сервером или же полученное от сервера в результате обработки запроса.
Класс PmsCertList Основные поля: X509Certificate2[] Certs; public string ErrStr;	Член Certs представляет собой массив сертификатов в стандарте X509. Класс содержащего несколько конструкторов для загрузки сертификатов из системных хранилищ (с поиском по имени владельца или серийному номеру) или из файлов. Член ErrStr содержит последнее диагностическое сообщение, возникшее при неудачной попытке загрузки сертификатов.

Организация сервера PMS основана на следующих принципах.

- Сервер PMS основан на мульти-потоковой обработке: для обслуживания каждого информационного запроса создается отдельная обрабатывающая программная нить (поток), в рамках которой организуется выполнение вызванной обра-

батывающей функции. Сервер берет на себя всю работу, связанную с подачей полученного по сети сообщения на вход функции, отправке результатов ее выполнения в сеть и отслеживания таймаута выполнения.

- Все множество сервисных обрабатывающих функций реализуется в форме одной или нескольких динамических библиотек. Никакие специальные «конструкции» типа Web-сервисов не используются. Поиск и загрузка всех динамических библиотек выполняются при запуске сервера PMS из каталога, указанного в конфигурации сервера (см. рис. 3). В каждой найденной библиотеке проводятся поиск и подключение всех функций, имеющих строго определенную спецификацию:

  PmsMessage *имя_функции* (PmsMessage Inpt,
  string [] Conf, ref string ErrMsg),

  в которой Inpt - входное сообщение (запрос); Conf – конфигурационные данные в форме массива строк вида «имя=значение»; ErrMsg – возвращаемое диагностическое сообщение. С этого момента все библиотечные функции, соответствующие данной спецификации, начинают играть роль сервисных функций, доступных для клиентских программ. Все остальные функции попросту игнорируются.
- При загрузке каждой динамической библиотеки сервер PMS выполняет поиск её конфигурационного файла, совпадающего по имени с файлом библиотеки, но имеющего расширение имени «.cfg» (своего рода аналог файла web.config для Web-сервисов). Этот файл может содержать определения строковых конфигурационных параметров в форме «имя=значение», как для библиотеки в целом, так и для каждой функции индивидуально. Конфигурационные данные сохраняются во внутренних структурах данных сервера и автоматически подаются в качестве значения параметра Conf на вход каждой функции при обращении к ней. Отметим, что некоторые важные системные параметры (например, таймаут выполнения) можно определять для каждой функции в отдельности (в отличие от Web-сервисов).

- В условиях строго одинаковой спецификации всех обрабатывающих функций применение WSDL практически теряет смысл. Вместо этого в PMS сделан акцент на обеспечение возможности отладки клиентских и сервисных компонент информационной системы (и их взаимодействия) вне сетевой среды на локальном компьютере. Эта возможность реализована путем введения особых режимов работы в метод Connect класса PmsConnection и в метод Process класса PmsMessage (см. рис. 1 и 2). Если при вызове метода Connect для какого-либо объекта типа PmsConnection ему передается не имя сервера PMS, а полная спецификация каталога, то вместо установления соединения с сервером выполняется загрузка динамических библиотек из этого каталога и подключение их к программе клиента (точно по тем же правилам, что и в сервере PMS). После этого выполнение метода Process с использованием этого объекта приведет попросту к вызову библиотечной функции в рамках клиентского процесса (см. рис. 4) с возможностью применения всех отладочных средств, имеющихся в среде разработки. Важно подчеркнуть, что отлаженные компоненты могут быть перенесены в продуктивную сетевую среду без каких-либо изменений.
- PMS в полной мере использует двоичную природу TCP/IP [4]. Взаимодействие между клиентом и сервером PMS осуществляется по специальному, достаточно простому протоколу, ориентированному на передачу двоичных сетевых сообщений в обоих направлениях. Каждое сообщение в общем случае содержит два массива байтов: заголовок сообщения и тело сообщения. Первые 4 байта заголовка или тела содержат целое число – его длину. При передаче запроса от клиента к серверу в заголовок сетевого сообщения помещается строка, содержащая полное имя вызываемой функции в формате «имя_библиотеки**.**имя_функции», а в тело сообщения упаковывается структура PmsMessage в открытой или зашифрованной форме, содержащая информационный запрос. Строка заголовка используется сервером для организации вызова соответствующей обрабатывающей функции.

При передаче результата обработки от сервера к клиенту в заголовок сетевого сообщения помещается строка диагностического сообщения (значение параметра ErrMsg, сформированное обрабатывающей функцией), а в тело сообщения упаковывается структура PmsMessage, содержащая ответ сервера в открытой или зашифрованной форме. Полученное от сервера диагностическое сообщение автоматически присваивается члену ErrStr объекта класса PmsConnection на стороне клиента.

- Главную нагрузку по обеспечению информационной безопасности берет на себя сервер PMS, а не обрабатывающие функции. В частности, он реализует дешифровку информационного запроса перед его передачей на вход обрабатывающей функции (если запрос зашифрован) и шифрование результата обработки перед его отправкой в сеть. Кроме того, на сервер можно опционально возложить функции проверки ЭЦП у входящих сообщений и добавления собственной ЭЦП к исходящим. Важно подчеркнуть, что проверка ЭЦП на сервере может включать не только аутентификацию информационного запроса, но и разграничение доступа к обрабатывающим функциям путем проверки реквизитов подписантов (имя, организация, должность и т.п.) по имеющимся в конфигурационных данных спискам доступа. Важно подчеркнуть, что такие списки доступа (как и таймаут выполнения) можно определять для каждой функции в отдельности и/или для всей библиотеки функций.

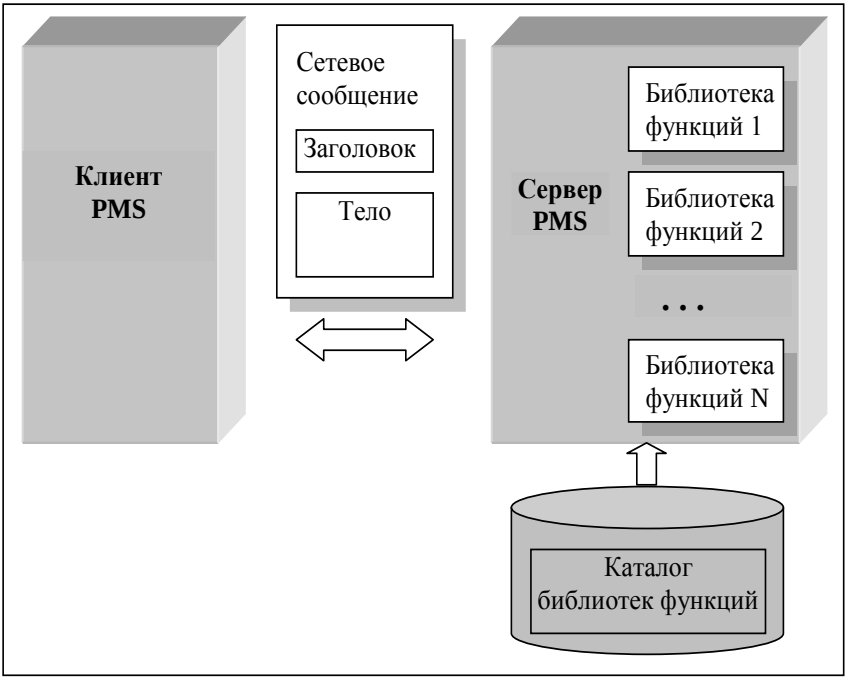

Рис. 3. Принципы организации сервера PMS

Ввиду того, что и клиентская библиотека PmsBase.dll и сервер PMS выполняют функции защиты и аутентификации данных, конкретная реализация службы должна опираться на определенную криптосистему (или на нескольких криптосистем с возможностью выбора одной из них). От применяемой криптосистемы, в частности зависят структура и алгоритмы формирования ЭЦП (т.е. класса Signature) и ключевой информации, содержащейся в шифрованном сообщении.

Для выполнения лабораторных экспериментов и получения сравнительных оценок быстродействия была проведена реализация службы PMS в среде программирования Microsoft VisualStudio 2010 для платформы .Net Framework 4.0 на основе криптосистемы «КриптоПро CSP» версии 3.6, соответствующей требованиям действующих в России ГОСТов в области криптографической защиты информации.

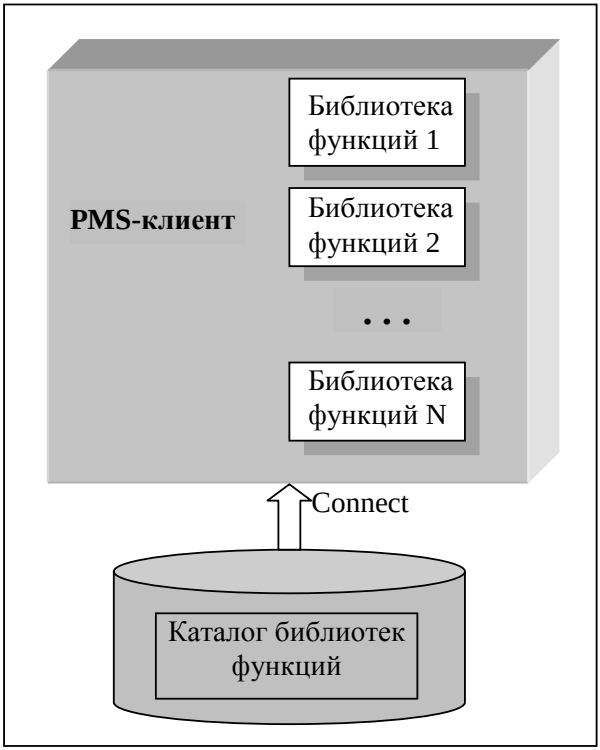

Рис. 4. Отладка сервисных функций-обработчиков вне сетевой среды

4. Временные оценки

Основная цель экспериментов с PMS заключалась в сравнении ее быстродействия с быстродействием Web-сервисов в одинаковых условиях. Главное внимание уделялось вызовам сервисных функций с относительно малым (от нескольких миллисекунд до нескольких сотен миллисекунд) временем выполнения (при более длительной обработке разница между двумя технологиями практически нивелируется) с применением средств криптозащиты и без применения. В экспериментах с

PMS использовались средства криптозащиты, интегрированные в клиентскую библиотеку и сервер PMS. В экспериментах с Web-сервисами средства криптосистемы «КриптоПро CSP» подключалась непосредственно к программам клиента и Web-сервиса. И сервер PMS с модельными библиотечными функциями и Internet Information Server (IIS) с модельными Web-сервисами были установлены на одном и том же четырехъядерном сервере приложений с тактовой частотой 2.4.ГГц в операционной среде Window 2008 Server, а в качестве клиентской рабочей станции использовался одноядерный компьютер с тактовой частотой 2.8 ГГц. Тестирование проводилось в скоростной (100 Мбит/сек.) локальной сети.

На рис. 5, 6 и 7 показаны характерные результаты экспериментов с очень быстрой сервисной функцией, выполняющей простое перекодирование полученного строчного сообщения в верхний регистр и возврат результата клиенту, при длине сообщения в 2 Кбайт, 50 Кбайт и 100 Кбайт соответственно. На каждом рисунке приведены диаграммы времен выполнения операции на сервере (в миллисекундах) с помощью Web-сервиса (черный столбик) и с помощью PMS (серый столбик) для трех режимов: без применения криптозащиты, с применением ЭЦП и с применением ЭЦП и шифрации сообщений. В каждом режиме время выполнения вычислялось, как среднее значение для 100 последовательных вызовов сервисной функции.

Как видно из рисунков, в данной серии экспериментов PMS не уступает технологии Web-сервисов в быстродействии и даже несколько превосходит её. Причем, в режиме без применения криптозащиты это превосходство является весьма значительным (что, по-видимому, целиком объясняется временными затратами, связанными с применением протокола HTTP/SOAP). При подключении криптозащиты время обработки возрастает и различие становится менее существенным (характерно, что абсолютная разница во времени выполнения при этом остается относительно стабильной).

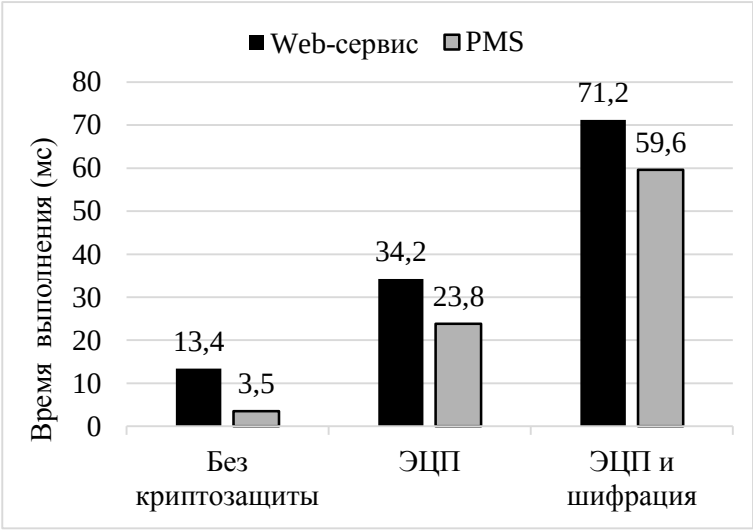

Рис. 5. Время обработки при длине сообщения 2 Кбайт

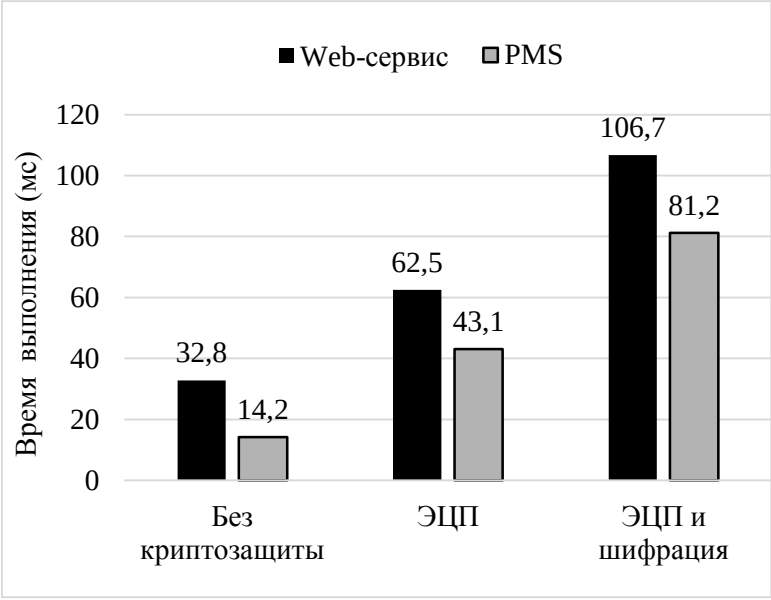

Рис. 6. Время обработки при длине сообщения 50 Кбайт

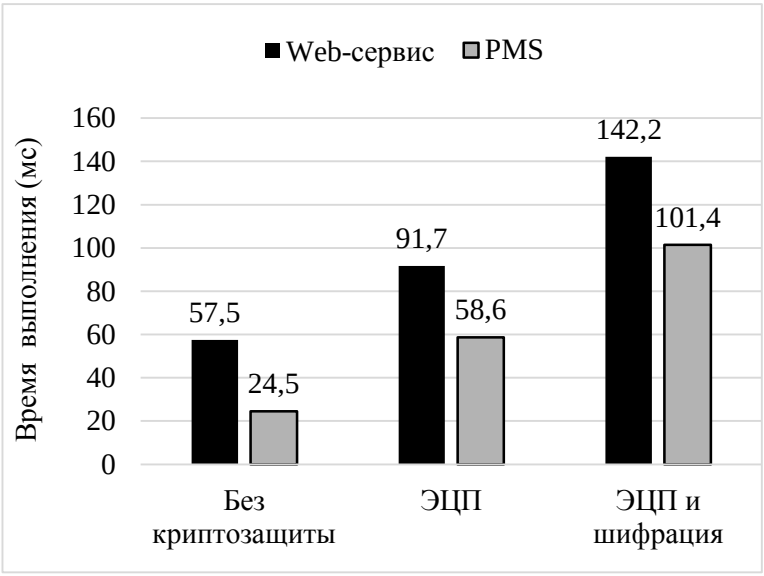

Рис. 7. Время обработки при длине сообщения 100 Кбайт

На рис 8 приведены результаты экспериментов с относительно медленными сервисными функциями со временем выполнения несколько сотен миллисекунд. Главное внимание в этой серии экспериментов уделялось не оценке времени выполнения одиночных запросов, но сравнению быстродействия технологий PMS и Web-сервисов в условиях высокой нагрузки: при одновременной обработке пакета информационных запросов на сервере. Скорость обработки вычислялась, как частное от деления числа запросов в пакете на полное время его выполнения. Характерные кривые, приведенные на рис.8, отражают зависимость скорости обработки от количества запросов в пакете для технологий PMS и Web-сервисов при обращении к модельным сервисным функциям со временами выполнения 0.5 и 2 секунды с применением средств криптозащиты и длине входного и выходного сообщений 50 Кбайт.

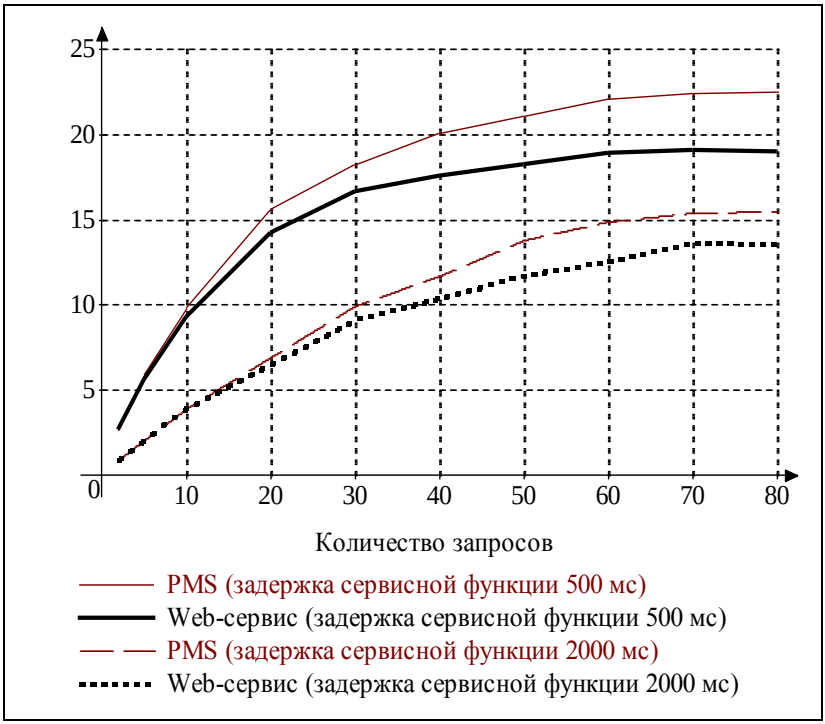

Рис. 8. Скорость обработки пакета одновременных запросов

Как видно из рисунка, все кривые ведут себя в целом одинаково. При увеличении числа запросов в пакете растет и скорость обработки, что объясняется положительным эффектом от многопоточной обработки запросов и в IIS и в сервере PMS. Например, при 20-ти запросах в пакете и времени выполнения сервисной функции 500 мс скорость обработки достигает 15.5 запросов в секунду у PMS и 14.2 – у Web-сервиса (очевидно, что при последовательной обработке скорость не могла бы превысить значения 2). Однако с дальнейшим увеличением размеров пакета рост скорости обработки замедляется, а потом и вовсе останавливается вследствие достижения предельной производительности. Как видно из графиков, в этой серии экспериментов PMS несколько превышает Web-сервисы по скорости обработки, но это превышение не является значительным.

В целом результаты экспериментов позволяют сформулировать следующие выводы.

- PMS в целом не уступает в быстродействии Web-сервисам (как при использовании средств криптозащиты данных, так и без использования).
- Применение средств криптозащиты в PMS не разрушает положительного эффекта от многопоточной обработки запросов.
- Как и Web-сервисы, PMS вполне позволяет поддерживать скорость обработки до нескольких и даже нескольких десятков запросов в секунду даже при использовании средств криптозащиты, что обычно бывает достаточным для большинства информационно-управляющих систем.

5. Заключение

Разумеется, сегодня ни одна сетевая технология не может сравниться с технологией Web-сервисов по гибкости и возможности применения в самых различных областях, связанных с распределенной обработкой. Тем не менее, при создании информационно-управляющих систем довольно часто встречается ситуация, когда готовые решения в области защиты и аутентификации информационных запросов в сети оказываются более важными для разработчика, чем возможность определения сервисных функций с любым количеством и любыми типами параметров.

В таблице 2 сопоставлены основные преимущества и недостатки Web-сервисов в технологии NET и PMS. Как видно из таблицы, только высокая скорость обработки является общим свойством двух технологий, в части же остальных характеристик списки преимуществ и недостатков зеркально противоположны. Подводя итог, можно заключить, что область эффективного применения PMS включает распределенные информационные системы, все компоненты которых разрабатываются в рамках одного проекта со строгой координацией и

администрированием (например, ведомственные системы или системы предприятий, имеющих региональные филиалы). Именно в таких разработках могут быть использованы такие преимущества PMS, как встроенные средства криптозащиты или возможность совместной отладки клиентских и серверных компонент вне сетевой среды. Наоборот, в системах с независимой разработкой отдельных компонент или подсистем (межведомственных, транснациональных и т.п.) гибкость в спецификациях сервисных функций и платформенная независимость Web-сервисов оказываются наиболее востребованными.

Таблица 2. Основные преимущества и недостатки технологий Web-сервисов и PMS

Технология	Преимущества	Недостатки
Web-сервисы	• Высокая гибкость в спецификациях сервисных функций. • Высокая скорость обработки. • Языковая и платформенная независимость.	• Избыточный сетевой трафик, связанный с упаковкой двоичных данных и XML-документов в структуру HTTP/SOAP. • Отсутствие собственных встроенных средств криптозащиты (кроме использования HTTPS с вытекающими отсюда ограничениями). • Затрудненность отладки сервисных компонент вне сетевой среды

PMS	• Высокая скорость обработки. • Тесная интеграция средств сетевого взаимодействия и средств криптозащиты. • Возможность отладки клиентских и сервисных компонент и их взаимодействия вне сетевой среды.	• Жесткость спецификации сервисных функций (строгая ориентация на модель обмена сообщениями). • Платформенная зависимость (Windows, .NET Framework 4.0). • Привязанность к определенной криптосистеме или группе криптосистем.

Литература

1. АСРАТЯН Р.Э., ЛЕБЕДЕВ В.Н. *Защита электронных сообщений в мульти-сетевой среде* // Управление большими системами. – 2013. – Выпуск 45. – С. 95–111.
2. ЗГОБА А.И.,МАРКЕЛОВ Д.В.,СМИРНОВ П.И. *Кибербезопасность: угрозы, вызовы, решения* // Вопросы кибербезопасности. – 2014. – № 5. – С. 30–38.
3. МАК-ДОНАЛЬД М.,ШПУШТА М. *Microsoft ASP.NET 3.5 с примерами на C# 2008 и Silverlight 2 для профессионалов*. – М.: Вильямс, 2009. – 1408с.
4. СНЕЙДЕРС Й. Эффективное программирование TCP/IP. Библиотека программиста. – СПб.: Символ-Плюс, 2002. – 320 с.
5. ХАНТ К. *TCP/IP. Сетевое администрирование.* – СПб.: Питер, 2007. – 816 с.
6. ШАПОШНИКОВ И.В. *Web-сервисы Microsoft .NET*. – СПб: БХВ-Петербург, 2002. – 336 с.
7. ЩЕГЛОВ А.В. *Защита компьютерной информации от несанкционированного доступа.* – СПб.: Наука и техника, 2004. – 384 с.

8. WANG V., SALIM F., MOSKOVITS P. *The definitive guide to HTML5 WebSockets.* – NY: Apress, 2013. – 208 p.

INTERNET SERVICE FOR PROTECTED INFORMATION QUERIES PROCESSING

Ruben Asratian, Institute of Control Sciences of RAS, Moscow, Cand.Sc., assistant professor (rea@ipu.ru).

Abstract: *The principles of the organization of the new network service intended for protected queries processing in the distributed information and control systems are considered. Distinctive features of service are the high speed of processing, close integration of authentication and data protection functions with functions of network information exchange and a possibility of debugging both client, and server components of system out of the network environment.*

Keywords: distributed systems, Web-technologies, Internet-technologies, network interactions, data security.

Статья представлена к публикации
членом редакционной коллегии ...заполняется редактором…

Поступила в редакцию ...заполняется редактором…
Опубликована ...заполняется редактором…