Лекция
по учебной дисциплине
"Сервис - ориентированная архитектура приложений"

Тема № 3 "Применение технологий и протоколов информационных систем"

I. Учебные цели

1. Дать обучающимся сведения об основных архитектурных принципах построения программного обеспечения, а также протоколов и форматов данных взаимодействия информационных систем.

II. Воспитательные цели

1. Воспитывать интерес к дисциплине, стремление совершенствовать свои знания и уверенность в приобретенных знаниях.
2. Стимулировать у обучающихся активную познавательную деятельность, способствовать формированию у них творческого мышления.

III. Расчет учебного времени

Содержание и порядок проведения занятия:
Вступительная часть – 10 мин.
Основная часть – 75 мин.
Учебные вопросы:
1. Определение и функционал клиента и сервера – 10 мин.
2. 2х и 3х -звенные архитектуры построения приложений – 15 мин.
3. Форматы данных и их использование – 50 мин.

Заключительная часть – 5 мин.

IV. Литература

1. Вольфсон, Михаил Борисович Управление ИТ- сервисами в гуманитарной сфере : [Электронный ресурс] : учебное пособие / М. Б. Вольфсон, Е. П. Охинченко, Н. Б. Андреева ; рец.: Ю. П. Левчук, А. А. Степаненко ; Федеральное агентство связи, С.-Петерб. гос. ун-т телекоммуникаций им. проф. М. А. Бонч-Бруевича. - СПб. : СПбГУТ, 2014. - 98 с. : ил. - 465.58 р. URL: http://lib.spbgut.ru/jirbis2_spbgut/components/com_irbis/pdf_view/?241211
2. Хабибуллин, И. Самоучитель XML : [Электронный ресурс] / И. Хабибуллин. - Санкт-Петербург : БХВ-Петербург, 2015. - 336 с. : ил. - URL: http://ibooks.ru/reading.php?productid=18392. - ISBN 978-5-9775-1973-1 : Б. ц.

V. Учебно-материальное обеспечение

1. Проектор, материал презентации.
2. Доска, стираемый маркер.

Введение

Мы продолжаем изучение архитектурных принципов построения программного обеспечения. В рамках лекции буду рассмотрены определение и функционал клиента и сервера, 2х и 3х -звенные архитектуры построения приложений а также форматы данных и примеры их использования.

Учебные вопросы

1. Определение и функционал клиента и сервера

Рассмотрим определения клиента и сервера на основе действующего ГОСТ Р ИСО/МЭК 10031-2000 "Информационная технология. Текстовые и учрежденческие системы. Модель приложений распределенного учреждения. Часть 1. Общая модель":

Клиент – часть прикладной системы, обеспечивающая интерфейс пользователя к серверу.

Сервер – часть прикладной системы, обеспечивающая функциональные возможности, специфицированные в определении предоставляемых им услуг.

Система серверов – совокупность одного или нескольких серверов, обеспечивающая функциональные возможности, специфицированные в определении предоставляемых ими услуг.

Под **протоколом** предметной области взаимосвязи открытых систем, в соответствии с ГОСТ Р 52292-2004 "Информационная технология. Электронный обмен информацией. Термины и определения", понимается набор семантических и синтаксических правил, определяющий поведение объекта на данном уровне при выполнении коммуникационных функций.

Реализация протоколов управления ресурсами осуществляется в автоматизированных системах и телекоммуникационном оборудовании.

Автоматизированная система (АС), это такая система, в которой часть выполняемых ею функций осуществляется без необходимости выполнения каких-либо действий со стороны персонала её эксплуатирующего. Под автоматизированной системой, в соответствии с ГОСТ 34.003-90 "Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Термины и определения", понимается система, **состоящая из персонала и комплекса средств автоматизации его деятельности**, реализующая информационную технологию выполнения установленных функций. Под комплексом средств автоматизации понимается совокупность всех компонентов автоматизированной системы за исключением персонала. Под информационной технологией понимаются приемы, способы и методы применения средств вычислительной техники (СВТ) при выполнении функций сбора, хранения, обработки, передачи и использования данных.

Классификация программных средств автоматизированных систем управления

Классификация программных средств автоматизированных систем управления приведена на основе выделения следующих функциональных, информационных и структурных классификационных признаков:

1) **По интероперабельности (унификации функций) управления**:

1.1) интероперабельные средства управления телекоммуникационным оборудованием, позволяющие тем или иным образом осуществлять управление телекоммуникационным оборудованием различных производителей, типов, видов, классов, назначения, технологий сред передачи и пр.;

1.2) средства управления телекоммуникационным оборудованием, позволяющие осуществлять управление телекоммуникационным оборудованием одного типа/вида и/или одного производителя.

2) **По количеству функций управления**:

2.1) многофункциональные средства управления телекоммуникационным оборудованием, реализующие в том или ином объеме функционал сбора, хранения, передачи, анализа и преобразования (обработки) информации состояния и формирования управляющей информации;

2.2) однофункциональные (монофункциональные) средства управления телекоммуникационным оборудованием, например, только функционал сбора в части запроса и сохранения первичной информации состояния без её анализа и обработки.

3) По виду лицензионных ограничений :

3.1) открытые средства управления телекоммуникационным оборудованием, распространяемые на основе простой (неисключительной) лицензии для свободно распространяемого программного обеспечения, которые, при необходимости, могут быть доработаны не запрещенными законом способами без нарушения лицензионной и/или патентной

чистоты (средства мониторинга и управления http://www.nagios.org, http://www.isc.org, http://www.shrubbery.net/rancid, http://cacti.net, http://opendcim.org, http://sourceforge.net/projects, http://www.balabit.com/network-security/syslog-ng, http://www.rsyslog.com и др.);

3.2) проприетарные (закрытые) средства управления телекоммуникационным оборудованием, распространяемые на основе простой (неисключительной) или исключительной лицензии, ограничивающей их использование и/или внесение изменений и/или распространение и/или накладывающей другие ограничения (средства мониторинга и управления WANDL Inc. (IP/MPLS View, Network Planning & Analysis Tools, Multilayer Network Design solution), Hewlett-Packard Development Company, L.P. (OpenView, Service Activator), IBM Corporation (Tivoli Netcool), Axiros GmbH (Generic Device Management, IP Address Management And Provisioning), Juniper Networks, Inc. (Network and Security Manager, Service and Resource Control-Policy Engine, Junos Space Service Now, Junos Space Service Insight, NetScreen product, NorthStar controller), Cisco Systems, Inc. (Network Management System, Unified Computing System), Huawei Technologies Co., Ltd. (iManager U2000, iManager N2510), D-Link Ltd. (D-View) и др.).

4) **По сложности решаемых функциональных задач в контуре управления телекоммуникационным оборудованием**:

4.1) средства, осуществляющие решение задач мониторинга ("почему так"), сбор информации состояния и его первичный анализ и обработка, реализованные как ТУ оборудованием "как есть";

4.2) средства, осуществляющие решение задач анализа информации состояния и моделирование возможных вариантов функционирования объектов учёта ("что будет, если") с целью решения задач ОТУ, включая средства управления , имеющие лицензионные ограничения, не позволяющие учитывать так называемое "виртуальное" оборудование и работающие только с активным физическим оборудованием;

4.3) средства, осуществляющие решение задач структурно-параметрического синтеза ("как сделать, чтобы") соответствующих сегментов сети (ОТУ и, частично, ОУ) по заданным исходным данным в виде рядов значений и/или вида входной информации (метрик, значений максимальных фактических задержек для сообщений каждого класса, цены за заданный объем переданных сообщений и пр.).

5) **По используемым протоколам управления телекоммуникационным оборудованием**:

5.1) средства, использующие открытые (имеющие открытую спецификацию) протоколы, такие, как, например, Internet Engineering Task Force Request For Comments, Broadband Forum Technical Reports, Open Geo-spatial Consortium Specifications and Standards, Open Mobile Alliance Device Management Protocols, Institute of Electrical and Electronics Engineers industry Standards, World Wide Web Consortium Recommendations, Open Artwork System Interchange Standards, Open Networking Foundation и пр.;

5.2) средства, использующие модифицированные открытые протоколы, такие, как, например, Juniper Next-Hop Tunnel Binding, Cisco Vlan Trunking Protocol, Cisco Rapid Spanning Tree Protocol и пр.);

5.3) средства, использующие проприетарные (закрытые) протоколы, такие как, например, IBM System Network Architecture consisting of a protocol stack, Juniper NetScreen Gatekeeper Protocol, Juniper NetScreen Redundancy Protocol, Juniper Trivial Network Protocol, Cisco Hot Standby Router Protocol, Huawei Smart Link Protocol и пр.);

5.4) средства, использующие открытые и/или модифицированные открытые и/или проприетарные протоколы.

6) **По оперативности управления телекоммуникационным оборудованием**:

6.1) средства, выполняющие обработку информации и формирование управляющих воздействий в (информационном) режиме, при котором требования к времени взаимодействия с внешними по отношению к ним средствами и/или системами не регламентированы;

6.2) средства, выполняющие обработку информации и формирование управляющих воздействий в режиме "мягкого" реального времени, при котором, как правило, обеспечивается

взаимодействие с внешними по отношению к ним средствами и/или системами во времени, соизмеримом со скоростью протекания процессов во внешних средствах и/или системах;

6.3) средства, выполняющие обработку информации и формирование управляющих воздействий в режиме "жёсткого" реального времени, при котором гарантированно обеспечивается взаимодействие с внешними по отношению к ним средствами и/или системами во времени, соизмеримом со скоростью протекания процессов во внешних средствах и/или системах.

В настоящее время средства управления, осуществляющие решение задач синтеза ("как сделать, чтобы") в достаточной степени не реализованы ни у одного из производителей по ряду причин, основными из которых являются:

отсутствие достоверного математического и методического аппарата решения задач структурно-параметрического синтеза и/или высокая временная (вычислительная) сложность решения данной задачи средствами существующего математического и методического аппарата для практически важных значений параметров, характеризующих размерность решаемой задачи;

решение задач синтеза во всём мире принято осуществлять за отдельную плату высококвалифицированными архитекторами уровня производителей оборудования и/или системных интеграторов эвристическими способами, сводящимися в большинстве случаев к итеративному решению задач анализа ("что будет, если") с результатом в виде низкоуровневого и высокоуровневого дизайнов соответствующих функциональных сегментов ;

автоматизация процесса решения задачи синтеза архитекторами уровня производителей оборудования и/или системными интеграторами в лучшем случае выполняется только на уровне внутренних частных методик, которые сразу же становятся тем, что принято называть "ноу хау", т.е. конфиденциальной информацией этих организаций, причём даже выполнение описания в виде методик вступает в противоречие с желанием высококвалифицированных сотрудников сохранить рабочее место в условиях возможных форс-мажорных обстоятельств, таких как, например, изменение экономической ситуации и, в связи с этим, может не обладать достаточными полнотой, точностью и адекватностью.

Обобщенный сравнительный анализ возможностей перечисленных выше средств управления и используемых ими протоколов затруднителен тем, что в свободном доступе нет полных и достоверных описаний реализаций этих средств и протоколов, а также тем, что в модифицированные и в проприетарные протоколы разработчиками могут быть внесены изменения без уведомления. При этом следует отметить, что протокол и/или суперпозиция (совокупность) двух или нескольких протоколов, исходя из требуемого и фактического реализуемого ими функционала, могут быть использованы как средство спецификации, формализации и одновременно реализации контура управления системы управления. В свою очередь, использование закрытых протоколов позволяет фактически вводить новые стандарты, что увеличивает технологическую зависимость от зарубежных транснациональных корпораций и/или стран и, по сути, представляет собой не что иное, как попытку монополизации соответствующей области .

2. 2х и 3х -звенные архитектуры построения приложений

В настоящее время выделяют следующие виды приложений:

Приложения организационного управления. Такие приложения отражают актуальное состояние предметной области (ПрО) в любой момент времени. В ИС с такими приложениями преобладает режим оперативной обработки транзакций OLTP (OnLine Transaction Processing) (под транзакцией понимается неделимый набор операций с БД). Для систем OLTP характерен регулярный поток простых транзакций, оперативно отражающих изменения состояния ПрО. Важными требованиями в этих системах являются высокая производительность обработки транзакций и гарантированная доставка информации при удаленном доступе.

Приложения поддержки принятия решений (DSS – Decision Support System). Эти приложения обеспечивают с помощью сложных запросов отбор и анализ данных в различных аспектах: временном, географическом или по иным показателям. Они характеризуются тем, что извлекают данные из разнородных источников, включая неструктурированные, производят

многомерный анализ данных, включают обработку статистики и элементы прогностики и моделирования, в частности анализ "что, если". В общем случае в ИС с приложениями DSS преобладают сложные транзакции и аналитическая обработка. В этом классе приложений выделяются системы оперативной аналитической обработки OLAP (OnLine Analysis Processing), приложения аналитического анализа данных (Data Mining) и разного рода экспертные системы (системы поддержки принятия решений, основанные на знаниях).

Информационно-справочные приложения. Пользовательский интерфейс таких приложений часто строится на принципах гипертекста и представляет собой группу логически связанных текстовых и графических материалов. К этому типу как подтипы могут быть отнесены системы электронной документации, обучающие системы и географические информационные системы (GIS), а также системы, основанные на использовании гипертекстового представления информации и мультимедиа.

Приложения автоматизации документооборота. Современные системы автоматизации документооборота нацелены на перевод бумажных документов в электронный вид, обеспечение пользователей средствами индексирования и поиска документов. Приложения этого типа могут включать средства коллективной работы с документами, использовать электронную почту, электронные бланки, различные редакторы.

Корпоративные информационные системы могут строиться на основе различных архитектур, основными из которых являются:

многотерминальные централизованные вычислительные системы;
системы на основе локальной сети ПК (файл-серверные приложения);
системы с архитектурой клиент-сервер;
системы с распределенными вычислениями;
офисные системы;
системы на основе Internet/Intranet-технологий.

Влияние архитектуры на построение приложений заключается в том, что от выбранной архитектуры зависит распределение функциональных компонентов между средствами на автоматизированном рабочем месте пользователя (клиентская часть приложений) и сервером (серверами) приложений. Соответственно, разрабатываемые средства СМПО должны включать в себя именно тот набор функциональных компонентов, который определяется распределением функций обработки информации данной архитектуре.

Типовыми функциональными компонентами приложений, реализуемых средствами общего и специального программного обеспечения, являются:

PS (Presentation Services) - средства представления. Обеспечиваются средствами, принимающими ввод от пользователя и отображающим то, что сообщает ему компонент логики представления PL.

PL (Presentation Logic) - логика представления. Управляет взаимодействием между пользователем и ЭВМ. Обрабатывает действия пользователя по выбору альтернативы меню, по нажатию кнопки или при выборе элемента из списка.

BL (Business or Application Logic) - прикладная логика. Набор правил для принятия решений, вычислений и операций, которые должно выполнить приложение, т.е. собственно содержательная часть СМПО – программная реализация математических моделей и методов (алгоритмов) решения задач управления.

DL (Data Logic) – логика управления данными. Операции с базой данных, которые нужно выполнить для реализации прикладной логики управления данными.

DS (Data Services) – операции с базой данных. Действия системы управления базой данных, вызываемые для выполнения логики управления данными, такие как манипулирование данными, определения данных, фиксация или откат транзакций и т.п.

FS (File Services) – файловые операции. Дисковые операции чтения и записи данных, обычно являющиеся функциями операционной системы.

Распределение функциональных компонентов приложений между клиентской частью и серверами, демонстрирующее зависимость построения приложений от архитектуры ИС, приведено в табл. 1.

Таблица 1.

Распределение функциональных компонентов приложений в различных архитектурах

Вариант архитектуры	Распределение компонентов			Пример реализации
	Клиент	Сервер-1	Сервер-2	
Централизованная многотерминальная система	PS	PL, BL, DL, DS, FS	–	Сервер Sun с X-терминалами в среде ОС Solaris
Локальная сеть ПК с файл серверными приложениями	PS, PL, BL, DL, DS	FS	–	Локальная сеть ПК, программы на FoxPro, Clipper и др.
Удаленный доступ к данным на сервере БД	PS, PL, BL, DL	DS, FS	–	Система клиент-сервер с доступом ПК к серверу БД Informix
Удаленный доступ к БД с использованием хранимых процедур	PS, PL, DL	BL, DS, FS	–	Система клиент-сервер, доступ ПК к серверу ORACLE в среде SCO Unix
Удаленный доступ к БД с разделением логики приложения	PS, PL, BL, DL	BL, DL, DS, FS	–	Система клиент-сервер, доступ ПК к серверу ORACLE на Sun Solaris
Удаленное представление данных с доступом к Unix-системе	PS, PL	BL, DL, DS,FS	–	Сеть ПК/станций, приложения на мониторе транзакций и СУБД в Unix
Удаленное управление файл-серверным приложением в сети	PS	PL, BL, DL, DS	FS	Связь удаленных ПК с сервером доступа WinView в сети для работы с СУБД FoxPro
Многотерминальный сервер приложений для доступа к СУБД	PS	PL, BL, DL	DS, FS	Сервер приложений на SCO Unix, доступ терминалов к ORACLE на HP
3 звенная система на Unix с монитором транзакций	PS, PL	BL, DL	DS, FS	Сеть ПК, сервер приложений на TUXEDO и СУБД ORACLE в среде Solaris на Sun
3 звенная система с монитором транзакций и разделением логики	PS, PL, BL	BL, DL	DS, FS	Аналогично предыдущему, но контроль данных выполняется на клиентских узлах

Основным общим требованием к ПО АСУ является обеспечение комплексного автоматизированного решения информационно-расчетных задач на всех этапах управления,

включая проведение расчетов и моделирования с целью подготовки рекомендаций для принятия решений по управлению, а также при деятельности. При этом:

временные характеристики функционирования ПО в целом и его отдельных составных частей в различных системах должны обеспечивать выполнение общих требований к временам реализации циклов и процессов управления.

функционирование ПО и его составных частей должно быть устойчивым при искажении или отсутствии необходимых исходных данных, при сбоях вычислительных средств, а также обеспечивать сохранность введенной в систему информации.

ПО должно быть эргономичным, обеспечивать удобство работы с ним на основе применения эффективных средств организации диалога, средств диагностики и защиты от ошибок пользователей.

ПО должно иметь модульную структуру, обеспечивающую возможность его изменения (модификации, наращивания) путем замены, включения и выключения отдельных модулей, реализующих элементарные функции по обработке данных.

Более детальные функциональные и конструктивные требования к ПО формулируются при формировании соответствующих технических заданий. При этом наиболее сложными, ответственными и трудно формализуемыми задачами разработчиков являются организация и проведение формализации автоматизируемых процессов управления, а также организация разработки ПО.

3. Форматы данных и их использование

Протоколы информационного обмена и средства, реализующие цикл управления ресурсами сетей, составляют технологическое управление (ТУ), т.е. управление отдельно взятым сетевым элементом без формального учёта его взаимосвязей, а также мониторинг его состояния в автоматическом и/или в автоматизированном режиме для поддержания на требуемом уровне его характеристик. Для осуществления ТУ ресурсами сети используются протоколы Internet Control Message Protocol (ICMP) и Simple Network Management Protocol (SNMP).

Протокол Internet Control Message Protocol (ICMP), RFC792, 950 (1981 г., ...)

Протокол ICMP является протоколом сетевого уровня ЭМВОС и представляет собой отдельный протокол, инкапсулируемый в протокол сетевого уровня IP. По реализованным функциям (контроля доставки, передачи пакетов различного размера и пр.) сходен с протоколом транспортного уровня Transmission Control Protocol (TCP). В ICMP-протоколе реализована передача сообщений об ошибках и сообщений о возникновении существенных условий и/или особых ситуаций в ИКСС. В ходе работы некоторые ICMP-сообщения могут генерировать сообщения об ошибках, которые могут обрабатываться процессами прикладного уровня.

На рис. 1 показан формат ICMP-сообщения.

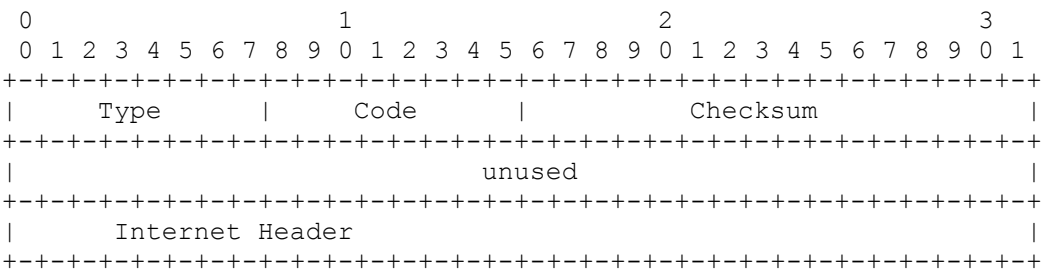

Рис. 1. Формат ICMP-сообщения

Существует 15 различных значений для поля типа (type), а для некоторых типов ICMP сообщений могут использоваться различные значения в поле кода (code).

Поле unused может не использоваться, а может и использоваться в зависимости от кода и типа ICMP-сообщений, например, для идентификации сообщения или для порядкового номера сообщения в серии ICMP-сообщений.

Рассмотрим некоторые, наиболее часто используемые значения типов ICMP-сообщений.

ICMP-сообщение о не достижимости конечной точки пункта назначения

```
Поля IP:
   Адрес назначения
Поля ICMP:
Тип
   3
Код
   0 = сеть не достижима
   1 = хост не достижим
   2 = протокол недостижим
   3 = порт недостижим
   4 = для передачи требуется фрагментация пакета, но установлен бит "пакет не фрагментировать"
   5 = ошибка маршрутизации
...
```

ICMP-сообщение об истечении времени ожидания

```
Поля IP:
   Адрес назначения
Поля ICMP:
Тип
   11
Код
   0 = время жизни пакета истекло при транзите
   1 = время повторной пересборки пакета истекло
...
```

ICMP-сообщения эхо-запросов и эхо-ответов

На рис. 2 приведен формат сообщения, в котором определены поля идентификатора и порядкового номера.

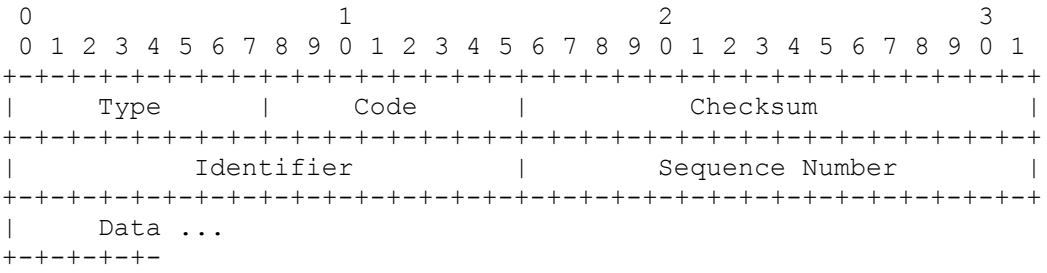

Рис. 2. Формат ICMP-сообщения эхо-запросов и эхо-ответов

```
Поля IP:
   Адрес назначения
Поля ICMP:
Тип
   8 для эхо-запросов
   0 для эхо-ответов
Код
   0
Идентификатор
   0
Порядковый номер
   0
```

Значения идентификатора и порядкового номера формально не определены и могут использоваться источником эхо-запросов произвольно. Вместе с тем, в эхо-ответе обязательно должны возвращаться те же самые значения идентификатора и порядкового номера, которые были получены в эхо-запросе. Идентификатор может совпадать с ID процесса, который формирует эхо-запрос, а может определять группу диагностируемых неисправностей для приложения прикладного уровня. В то же время порядковый номер может сохранять значение конкретного метода обработки приложения прикладного уровня соответствующих сообщений эхо-запроса.

ICMP-сообщения с запросом и получением меток времени

На рис. 3 приведен формат сообщения, добавлены поля метки времени отправителя, в котором определены метки времени получателя и времени передачи.

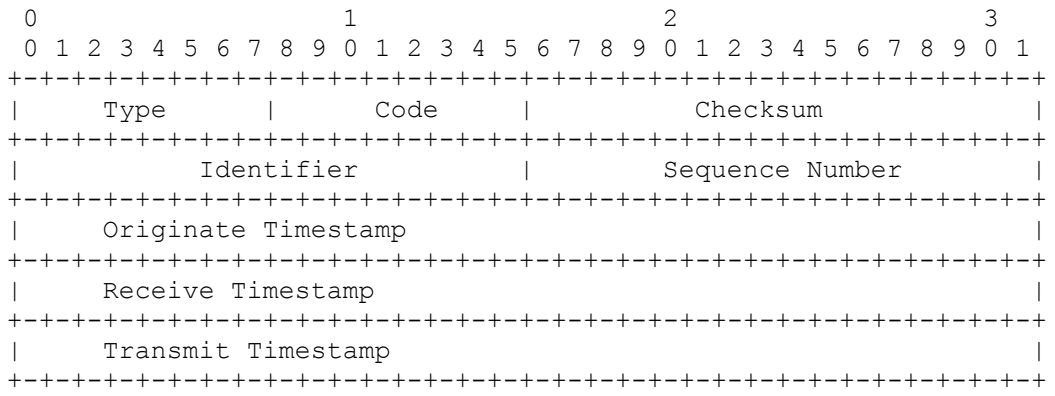

Рис. 3. Формат ICMP-сообщения с запросом и получением меток времени

```
Поля IP:
   Адрес назначения
Поля ICMP:
Тип
   13 для сообщения с запросом метки времени
   14 для сообщения с получением метки времени
Код
   0
Идентификатор
   0
Порядковый номер
   0
Временная метка отправления запроса запрашивающей системы
Временная метка текущего времени системы назначения
Временная метка системы назначения, затраченная на передачу ответа
```

ICMP-сообщения с запросом временной метки позволяет системе запросить другую систему о текущем времени. Значение, которое должно быть возвращено, это количество миллисекунд, которые прошли с полуночи в формате UTC, (Universal Coordinated Time, UTC универсальное скоординированное время). Особенность ICMP-сообщения с запросом временной метки заключается в том, что оно предоставляет время с точностью до миллисекунд, тогда как другие методы, используемые для получения времени от удаленных систем предоставляют время с точностью до секунд. Недостаток ICMP-сообщения с запросом временной метки заключается в том, что сообщается только количество секунд, прошедшее от начала текущих суток, т.е. запрашивающая система должна иметь информацию о текущей дате и временной зоне системы назначения.

Запрашивающая система заполняет исходную временную метку и отправляет запрос. Система назначения заполняет свое поле временной метки, когда получает запрос, и временную метку, затраченную на подготовку и передачу ответа. В большинстве программных реализаций оба последних поля принимают одно и то же значение.

Для синхронизации времени целесообразно использование специально разработанных для этого протокола NTP (v1: RFC1059, v2: 1119, v3: 1305, v4: 5905) или протокола SNTP (RFC1769, IPv4, IPv6: 4330).

Помимо рассмотренных, существует ряд других типов ICMP-сообщений. Подробно они рассмотрены в RFC792 и RFC950. Таким образом, протокол ICMP разработан для контроля ошибок в работе активных сетевых устройств и проверки работы механизмов контроля доставки пакетов различной длины, проверки связности систем и др. В настоящее время протокол ICMP используется для проверки доступности сетевых маршрутов и мониторинга доступности логических атрибутов соответствующего активного телекоммуникационного оборудования (ТКО).

Протокол Simple Network Management Protocol (SNMP), v1: RFC 1157, 1212, 1213, v2c: 1441-1452, 1901-1908, v3: 3411, 3418 (1990 г., ...)

Управление сетью с использованием протокола Simple Network Management Protocol (SNMP) основано на взаимодействии между программным обеспечением системы управления элементом ресурсов ИКСС ("менеджер") и, в большинстве случаев, микропрограммным обеспечением элементов ресурсов ИКСС ("агент"). Следует отдельно особо отметить, что программное обеспечение "менеджера" не всегда сложнее программного обеспечения "агента", т.е. программным обеспечением "агента" на стороне управляемого объекта для средств вычислительной техники является, например, сервер telnet, ssh или snmp (соответствующий "даемон" в терминах GNU Linux), а на стороне объекта управления, т.е. "менеджера" может быть простенькое программное приложение-клиент, функционирующее в командной строке, а может – достаточно высокопроизводительная система управления уровня производителя телекоммуникационного оборудования. Цикл управления заключатся в том, что "менеджер" обращается к "агенту" с запросом определенного значения (например, "сколько было сгенерировано ICMP ошибок о недоступности порта?") или с запросом на установку значения переменных в "агенте" ("изменить значение TTL по умолчанию на 64"). "Агент", в свою очередь, имеет возможность сообщить "менеджеру" о каком-либо важном произошедшем событии (SMNPtrap "подключенный интерфейс eth0 не работает").

SNMP использует два протокола и одно правило кодирования:

Structure of Management Information (SMI) – протокол задания правил формального описания объектов, присвоения имен и перечисления простых и структурированных типов с использованием абстрактной синтаксической нотации (Abstract Syntax Notation, ASN.1) на основе основных правил кодирования (Basic Encodes Rules, BER);

Management Information Base (MIB) – протокол формирования информационной базы об управляемых и контролируемых элементах ресурсов ИКСС с указанием соответствующих значений параметров (характеристик), их фактических количества, имен и типов.

Обмен данными, как правило, двусторонний и основан на известной концепции AV-пары: "атрибут" – "значение атрибута" для команд установки и получения соответствующего значения или просто "атрибут" для команды получения соответствующего значения. В SNMP в данную концепцию добавлены поле с типами данных, приведенными как в ASN.1, так и определенными непосредственно SNMP, а также метод кодирования передаваемых данных BER. Рассмотрение SMI, ASN.1 и BER выходит за рамки настоящей лекции и является предметом изучения разработчиков соответствующего программного и информационного обеспечений функционирования автоматизированных систем и ТКО.

В настоящее время существуют три наиболее распространенных спецификации SNMP-протокола – v1, v2c и v3.

Версия SNMP-протокола v1 изначально создавалась для управления ТКО, в котором отсутствовали требования по безопасному функционированию и управлению ТКО. Считалось, что создаваемый протокол v1 не работоспособен на СВТ и ТКО того времени (1990 г.). Развитие СВТ и ТКО практически опровергло это утверждение.

Версия SNMP-протокола v2c (1993 г.) является расширением версии v1 в части производительности и функциональности (ряд команд, которые будут рассмотрены далее). Расширения безопасности, которые были введены в версии v2, ввиду их высокой сложности, не получили внедрения и безопасность в реализации v2c осталась на уровне v1. В настоящее время для современного ТКО версия v2c является наиболее распространенной, а требования безопасности решаются другими техническими и организационными методами, которые выходят за рамки настоящей лекции.

Версия SNMP-протокола v3 (2002 г.) была разработана для расширения функционала и реализации требований по безопасному информационному обмену и управлению телекоммуникационным оборудованием средствами криптографической защиты. Ввиду достаточно сложной реализации и существенных требований к производительности данная версия до настоящего времени не получила широкого распространения в ТКО. Вместе с тем, данная версия реализована в общем программном обеспечении СВТ и в самом современном и высокопроизводительном микропрограммном обеспечении ТКО, в котором функционирует, по сути, операционная система GNU Linux (Juniper Inc., Free BSD и др.).

MIB-база, а точнее, MIB-II-база (версии 2) представляет собой дерево (связный ациклический граф), элементы которого либо являются ветвями с указанием имени, индекса, количества листов и ветвей, либо являются листьями с указанием имени, индекса, количества, типа и, возможно, значения по умолчанию. Пример части MIB-II-базы приведен на рис. 4.

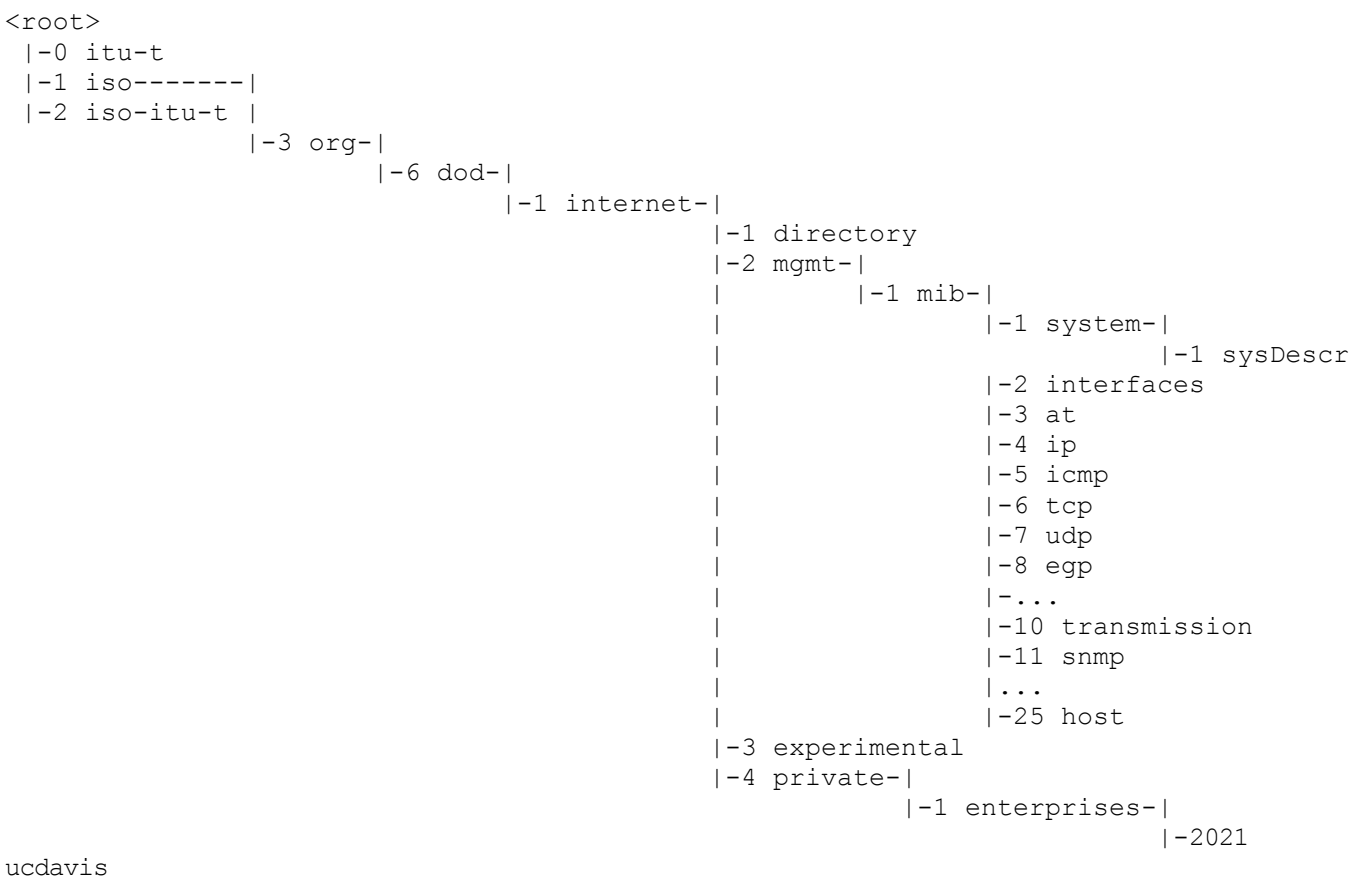

Рис. 4. MIB-II-база

Соответствующий лист или ветвь дерева MIB-II имеет идентификатор, который называется Object IDentificator (OID) и осуществляется через разделитель "точку" в двух функционально аналогичных видах – числовом и текстовом. Например, для элемента sysDescr:

.1.3.6.1.2.1.1.1

iso.org.dod.internet.mgmt.mib.system.sysDescr

Соответственно, для автоматизации удобно использовать числовую запись, а человеку-оператору удобнее использовать текстовую запись или, иными словами, то, что в литературе по эргономике называется "human-readable string".

Показанная на рис. 4 ветвь iso.org.dod.internet.private.enterprises (1.3.6.1.4.1) используется для создания MIB-баз различных производителей ТКО, СВТ и программного обеспечения, а имеющаяся в ней ветвь iso.org.dod.internet.private.enterprises.ucdavis (1.3.6.1.4.1.2021) – для самостоятельного создания необходимого OID системным администратором средствами системного администрирования. Следует отметить, что программное обеспечение, осуществляющее обслуживание ветви iso.org.dod.internet.private.enterprises.ucdavis (1.3.6.1.4.1.2021) должно обладать требуемыми производительностью, устойчивостью, надежностью и безопасностью, чтобы не нарушить эти характеристики для основного "даемона" SNMP.

На рис. 5 показан состав ветви iso.org.dod.internet.mgmt.mib.udp (.1.3.6.1.2.1.7).

```
...
 |-7 udp-|
         |-1 udpInDatagrams
         |-2 udpNoPort
         |-3 udpInErrors
         |-4 udpOutDatagrams
         |-5 udpTable-|
                      |-1 udpEntry-|
                                   |-1 udpLocalAddress
                                   |-2 udpLocalPort
```

Рис. 5 Состав ветви iso.org.dod.internet.mgmt.mib.udp

Для получения доступа к листу ветви iso.org.dod.internet.mgmt.mib.udp следует получить полный путь к соответствующему OID и передать его на вход протоокольной единице обмена GetRequest. Полный путь состоит из пути к ветви

.1.3.6.1.2.1.7

номера листа, например, для udpOutDatagrams

.4

и номера экземпляра листа, нумерация экземпляров листа начинается с 0

.0

Результат:

.1.3.6.1.2.1.7.4.0

iso.org.dod.internet.mgmt.mib.udp.udpOutDatagrams.0

Получение доступа к элементу udpEntry, представляющему собой таблицу, несколько сложнее, т.к. составляющие её элементы могут не начинаться с 0. Вместе с тем, зная OID до элемента udpEntry можно осуществить чтение всего, что находится за ним с использованием протокольной единицы обмена GetNextRequest, указав соответственно .1.3.6.1.2.1.7.5 или .1.3.6.1.2.1.7.5.1. В результате будет получена таблица элемента udpEntry с соответствующими OID и их значениями. Возможны и другие способы получения как части, так и всего дерева MIB.

Рассмотрим протокольные единицы обмена (Protocol Data Units, PDU) или, как их достаточно часто не формально называют, команды SNMP-протокола применительно к версиям v1, v2c и v3:

GetRequest – получить значение одного или нескольких OID (v1, сокет 161)
GetNextRequest – получить значение следующего OID после указанного OID (v1, сокет 161)
GetSubtree – получить значение всех OID данного поддерева, начиная от заданного (v1, сокет 161)
Walk – получить значение всех OID, начиная с заданного (v1, сокет 161)
Response – переслать значения одного или нескольких запрошенных OID (v1, сокет 161)
SetRequest – установить значение одного или нескольких OID (v1, сокет 161)
Trap – переслать значения одного или нескольких OID в связи с изменением их значений (v1, сокет **162**)

GetBulkRequest – получить значение одного или нескольких OID, применяется для большого количества блоков данных в одном запросе вместо итеративного применения GetNextRequest(v2, сокет 161)
InformRequest – передать данные от "менеджера" к "менеджеру" с подтверждением получения (для их обработки или, возможно, последующего проксирования во внешнюю систему управления) (v2, сокет 161)

Report – передать информацию о возможных ошибках при передаче данных от "менеджера" к "менеджеру"(v3, сокет 161)

Формат пакетов передаваемых и принимаемых протокольных единиц обмена SNMP-протокола различен в каждой из рассматриваемых версий и между собой они не совместимы. Вместе с тем, "даемон" SNMP, при надлежащих настройках его конфигурации, одновременно может поддерживать две или все три версии.

Следует отметить, что протоколы ICMP и SNMP могут использоваться не только для управления отдельно взятым сетевым элементом ресурсов ИКСС, но и для предоставления услуг и мониторинга сетей, составляющих ИКСС. Реализация предоставления услуг с использованием ICMP практически не возможна, т.к. он создавался только для проверки сетевой доступности, т.е., на его основе можно реализовать услугу мониторинга. Использование протокола SNMP для конфигурирования услуг затруднительно из-за наличия различных MIB-баз для каждого соответствующего ТКО и, в связи с этим, протокол SNMP используется для мониторинга доступности и сбора статистической информации в основном, в части стандартизованных общеупотребительных OID. Использование SNMP-протокола для конфигурирования услуг целесообразно при взаимодействии ТКО и специально разработанной для этого ТКО АС управления соответствующего производителя ТКО, классификация и некоторые примеры реализаций которых будут рассмотрены далее в вопросе № 3.

Протоколы управления ресурсами ИКСС для предоставления услуг осуществляются на уровне оперативно-технического управления (ОТУ), под которым понимается управление сетями и услугами связи в масштабе времени, соответствующем интенсивности возникновения ситуаций, требующих управляющих воздействий и времени исполнения этих воздействий, достаточном для функционирования ИКСС с требуемым качеством. В настоящее время для осуществления ОТУ ресурсами с целью конфигурирования услуг, предоставляемых ИКСС, в основном, используются протоколы telnet и ssh.

Протокол Teletype Network (telnet), RFC854, 855, 856, 857, 858, 859, 860, 885, 927, 930, 933, 946, 1041, 1053, 1073, 2196, 2197 (1983 г., ..., 1997 г., ...)

Telnet (Teletype Network, Terminals Network, telnet) – сетевой протокол удалённого доступа к активному экземпляру ресурса ИКСС с помощью командного интерпретатора. Это же название

имеет общее программное обеспечение различных операционных систем для работы с данным протоколом.

Протокол telnet позволяет обслуживающему СВТ рассматривать все удаленные терминалы как стандартные "сетевые виртуальные терминалы" строчного типа, работающие в коде ASCII, а также обеспечивает возможность согласования более сложных функций (например, локальный или удаленный эхо-контроль, страничный режим, высота и ширина экрана и т.д.) Протокол telnet работает на базе протокола TCP. На прикладном уровне над протоколом telnet на стороне пользователя находится программа-клиент поддержки реального терминала, а на стороне конфигурируемого СВТ – прикладной процесс ("даемон"), к которому осуществляется доступ с терминала пользователя.

В основу протокола telnet положены три принципа:

принцип виртуальных сетевых терминалов (Netvork Virtual Terminal, NVT) – после установления соединения предполагается, что каждый участник (и пользователь и "даемон") работает с виртуальным сетевым терминалом – мнимым устройством, выполняющим стандартные сетевые функции обычного терминала,

принцип настраиваемых параметров соединения, под которым понимается использование предоставляемых прикладным процессом ("даемоном") функций и/или дополнительных возможностей, которые клиент использует "на лету", без переинициализации соединения в случае, если он в состоянии их использовать,

принцип симметрии терминалов и процессов, под которым понимается равноправие участников соединения.

Виртуальный сетевой терминал NVT – устройство для ввода/вывода 7-и битных ASCII символов. Все преобразования и кодировки выполняются до NVT и не рассматриваются как часть NVT. NVT имеет устройство ввода "виртуальная клавиатура" и устройство вывода "виртуальный принтер", ввод и передача буферируются, данные накапливаются в буфере пока не будет завершена строка или не будет выполнена передача до завершения строки. Иногда, ввиду не полной совместимости терминалов, может осуществляться ввод символов команд без их отображения на экране, при этом сами команды после завершения строки (нажатия на Enter) будут выполнены и, возможно, результат их выполнения будет отображен на экране.

Для начала работы с удаленным терминалом пользователь должен выполнить аутентификацию, авторизацию результаты которых записываются в аккаунтинг. Иными словами он должен ввести правильную пару "имя пользователя "-"пароль пользователя" (login - password) (аутентификация), получить соответствующий имени пользователя набор прав для работы (авторизация) и информация о работе пользователя этом должна быть записана в систему учета (аккаунтинг).

После того как соединение по протоколу telnet установлено, начинаются переговоры об используемых опциях. Каждая из высоких договаривающихся сторон может послать другой один из четырех управляющих запросов с указанием соответствующей опции, если для этого у нее есть право:

WILL – а) запрос на запуск, б) согласование запроса на запуск

WONT – а) отказ запросу на запуск, б) запрос на отключение, в) согласование запроса на отключение

DO – а) согласование запроса на запуск, б) запрос на запуск

DONT – а) отказ запросу на запуск, б) согласование запроса на блокировку, в) запрос на отключение

Пример 1 (локальный запуск опции):

```
WILL "желаю запустить опцию 5" ->
                                        <- DO "запускай"
WILL "запускаю опцию 5" ->
```

В рассматриваемом примере вместо `<- DO "запускай"` могло быть `<- DONT "не запускай"`

Пример 2 (удаленный запуск опции):

```
DO "прошу запустить опцию 4" ->
                                        <- WILL "запущена, используй"
```

В рассматриваемом примере вместо `<- WILL "запущена,используй"` могло быть `<-WONT "не запущена"`

Пример 3 (локальное отключение опции):

```
WONT "желаю отключить опцию 2" ->
                                        <- DONT "отключай"
```

Пример 4 (удаленное отключение опции):

```
WONT "прошу отключить опцию 8" ->
                                        <- WONT "опция отключена/не используется"
```

Протокол telnet поддерживает работу в трех режимах:
заданный по умолчанию режим,
символьный режим,
режим строки (линии).

Режим, заданный по умолчанию, используется, когда с помощью опции переговоров не запрошены никакие другие режимы. В этом режиме возвращение символов делается клиентом. Пользователь печатает символ, а клиент отображает символ на экране (или принтере), но не посылает его, пока не закончится вся строка. После посылки полной строки на сервер клиент ждет команду GA (go ahead) от сервера, перед принятием новой строки — от пользователя. Эта работа — полудуплексная. Полудуплексная работа в настоящее время практически не используется, т.к. связь с использованием TCP является дуплексной.

В символьном режиме каждый напечатанный клиентом символ посылается серверу. Сервер обычно обрабатывает символ, чтобы отобразить на экране клиента. В этом режиме отображение символа может иметь продолжительное и заметное пользователю время (в случае если передача происходит, например, с использованием спутниковой связи) создает трафик в ИКСС, потому что для каждого символа данных нужно послать три сегмента TCP:
пользователь вводит символ, который посылает серверу;
сервер признает полученный символ и повторяет символ назад (в одном сегменте);
клиент подтверждает получение отображенного на экране символа.

Режим строки был предложен, чтобы компенсировать недостатки режима по умолчанию и символьного режима. В этом режиме редактирование строки (повторение, стирание символа, стирание строки и так далее) делается клиентом. Затем клиент посылает целую строку серверу. Режим строки напоминает режим, заданный по умолчанию, но это внешнее сходство. Режим, заданный по умолчанию, работает в полудуплексном режиме; режим строки является дуплексным, с клиентом, посылающим одну строку за другой, без потребности во вмешательстве символа GA (иди дальше — go ahead) от сервера.

Пример 5. В этом примере рассматривается работа режима по умолчанию и его недостатки Клиент и север ведут переговоры о типе терминала и скорости терминала, и затем сервер проверяет пароль пользователя.

Клиент	**Сервер**
	<- GO AHEAD
WILL терминал ->	
	<- DO терминал
	<- GO AHEAD
	<- WILL скорость терминала
	<- DONT скорость терминала
	<- LOGIN
	<- GO AHEAD

```
u ->
                                                    <- u
s ->
                                                    <- s
e ->
                                                    <- e
r->
                                                    <-r
n ->
                                                    <- n
a ->
                                                    <- a
m ->
                                                    <- m
e ->
                                                    <-e
                                                    <- PASSWORD
                                                    <- GO AHEAD
superpassword ->
                                                    <- GO AHEAD
ls –la ->
...
```

Пример 6. В этом примере клиент переключается в символьный режим и запрашивает сервер, чтобы запустить в работу опции SUPPRESS GO AHEAD и ECHO.

```
Клиент                                              Сервер
                                                    <- GO AHEAD
DO SUPPRESS GO AHEAD ->
                                                    <- WILL SUPPRESS GO AHEAD
DO ECHO
                                                    <- WILL ECHO
                                                    <- LOGIN
u ->
                                                    <- u
s ->
...
```

Рассмотренные примеры на практике скрыты от пользователя интерфейсом командной строки, который реализует перевод высокоуровневых команд пользователя в команды протокола. Высокоуровневыми командами пользователя являются:

`open` – связь с удаленным СВТ,
`close` – завершение связи,
`display` – отображение текущих рабочих параметров,
`mode` – изменение режима строки или символьного режима,
`set` – установка значений рабочих параметров,
`status` – отображение информации о состоянии,
`send` – передача специальных символов,
`quit` – выход из терминала telnet.

Протокол telnet не использует шифрование и уязвим для атак, поскольку вся информация может быть перехвачена нарушителем. Равную функциональность при б*о*льшей защищённости обеспечивает рассматриваемый далее протокол Secure Shell.

Протокол Secure SHell (ssh), v2:RFC4250-4256, 4335, 4344, 4345, 4419, 4432, 4716 (v1: 1995 г., v2: 2006 г.)

Протокол ssh (Secure SHell, ssh) – протокол, функционирующий на прикладном уровне ЭМВОС и позволяющий производить удалённое управление компьютером и передачу файлов. Функционально сходен с протоколом telnet, но использует криптографическую защиту передаваемой информации. Алгоритмы криптографической защиты протокола ssh не фиксированы и при установлении соединения возможен выбор различных алгоритмов шифрования. Протокол позволяет использовать безопасную удалённую оболочку командной строки на СВТ, но и туннелировать графический пользовательский интерфейс и любой другой сетевой протокол или сокет, обеспечивая, при соответствующем конфигурировании, возможность безопасной пересылки не только графики, но и, например, звука.

Версии протокола ssh.

Существует две версии протокола ssh – 1 и 2. Развитие версии 1 протокола ssh остановлено, так как в конце 1990-х в ней было найдено множество уязвимостей в криптографической защиты передаваемой информации, устранение которых было выполнено в версии 2.

Способы аутентификации клиента по протоколу ssh:

по ip-адресу клиента,

по публичному ключу клиента,

по паролю.

Этапы работы протокола ssh версии 2:

при запросе клиента сервер сообщает ему, какие методы аутентификации он поддерживает (определяется в опции PreferredAuthentications конфигурационного файла sshd.conf) и клиент по очереди пытается их использовать,

по умолчанию клиент вначале пытается аутентифицироваться своим адресом, затем публичным ключом и, если аутентификация не выполнена, передаёт пароль, введённый с клавиатуры (при этом пароль шифруется асимметрическим шифрованием),

после прохождения аутентификации одним из методов на основе имеющихся у клиента публичного и имеющего у сервера секретного пары ключей генерируется ключ симметрического шифрования, действующий в данной сессии,

затем клиенту назначаются права доступа к СВТ (авторизация) и информация об этом поступает в системное хранилище сообщений (аккаунтинг),

все последующие данные, передаваемые по протоколу ssh, шифруются данным ключом (обычно используется алгоритм aes с длиной ключа 128 бит).

Ключи хоста, используемые протоколом ssh.

Каждый хост должен иметь ключ. Хост могут иметь несколько ключей, используемых различными алгоритмами. Несколько хостов могут разделять один ключ. Каждый сервер должен иметь, по крайней мере, один ключ для каждого обязательного алгоритма открытого ключа. С помощью ключа сервера при обмене ключа можно проверить, действительно ли клиент общается с нужным сервером, для чего клиент должен предварительно знать об открытом ключе сервера. Могут использоваться две различные модели работы с ключами:

клиент имеет локальную базу данных, связывающую каждое имя сервера с соответствующим открытым ключом, что не требует централизованной административной инфраструктуры и трехсторонней координации. Это – наиболее часто используемая модель.

взаимосвязь "имя сервера – ключ" проверяется некоторым доверенным сертификационным органом – CA. Клиент знает только ключ корневого CA и может проверить достоверность всех ключей серверов, сертифицированных этими CA. Второй способ легче с точки зрения поддержки, так как у клиента необходимо хранить только ключ (единственный!) корневого CA. С другой стороны, каждый хост должен быть соответствующим образом сертифицирован перед тем, как можно будет установить безопасное соединение, т.е. должна быть проведена предварительная

работа по соответствующей настройке. Также нужно поддерживать работоспособность корневого CA.

Политики безопасности протокола ssh.

Протокол ssh позволяет вести переговоры об алгоритмах и форматах шифрования, целостности, обмена ключей, сжатия и открытого ключа. Алгоритмы шифрования, целостности, открытого ключа и сжатия могут отличаться для каждого направления или каждого порта, что должно быть решено указанием параметров конфигурации:

должны быть определены наиболее предпочтительные алгоритмы шифрования, целостности и сжатия для каждого направления и каждого порта,

должны быть определены используемые алгоритмы открытого ключа и метод обмена ключей для аутентификации сервера,

должны быть определены методы аутентификации, которые использует сервер для каждого пользователя. Может быть определено требование множественной аутентификации для некоторых или для всех пользователей. Методы и алгоритмы аутентификации могут также зависеть от того, с какого адреса пользователь пытается подключиться,

должны быть определены операции, которые пользователю разрешено выполнять, используя протокол соединения.

Способы вывода текста на терминал пользователя протоколом ssh.

В большинстве случаев протокол ssh непосредственно не выводит текст на терминал пользователя. Тем не менее, существует несколько случаев, в которых данные должны быть выведены на терминал и при этом набор символов должен быть указан явно (в большинстве случаев используется кодировка ISO 10646 UTF-8 и определяется поле для тега языка).

Существует известные трудности с набором символов для интерактивных сессий. Простого решения не существует, так как различные приложения могут показывать данные в различных форматах. Клиентом также могут применяться различные типы эмуляции терминалов, и используемый набор символов зависит от эмуляции терминала. Таким образом, для непосредственного описания используемого набора символов для данных терминальной сессии единственного варианта не существует. Тем не менее, стандартный тип эмуляции терминала "vt100" передается клиенту и неявно определяет набор символов и кодировку. Приложения обычно используют тип терминала для определения набора символов, или набор символов определяется с использованием внешних параметров.

Правила именования алгоритмов протокола ssh.

протокол ssh ссылается на конкретные алгоритмы хэширования, шифрования, целостности, сжатия и обмена ключа по именам. Существуют некоторые стандартные алгоритмы, которые должны поддерживать все реализации, а также алгоритмы, которые определены в протоколе, но являются дополнительными. Для имен алгоритмов существует два формата:

имена основных алгоритмов, которые не содержат символ "@", зарезервированы для обозначений IANA. Примерами являются '3des-cbc', 'sha-1', 'hmac-sha1'. Имена данного формата не должны использоваться без регистрации в IANA. Зарегистрированные имена не должны содержать символов "@" или ".".

имена дополнительных алгоритмов в формате name@domainname. Часть, следующая за символом @, должна представлять собой полностью определенное доменное Internet-имя лица или организации, его определяющая.

Состав подуровней протокола ssh:

протокол транспортного подуровня (ssh-trans) обеспечивает аутентификацию сервера, конфиденциальность и целостность соединения, может дополнительно обеспечивать сжатие данных, выполняется поверх соединения TCP, но может использоваться и поверх любого другого надежного соединения,

протокол аутентификации пользователя (ssh-userauth) аутентифицирует клиента для сервера и выполняется поверх протокола транспортного подуровня ssh-trans.

протокол соединения (ssh-conn) мультиплексирует несколько логических каналов в один зашифрованный туннель, выполняется поверх протокола аутентификации пользователя ssh-userauth. Пользователь посылает запрос всякий раз, когда устанавливается безопасное соединение на транспортном уровне. Второй запрос посылается после выполнения аутентификации пользователя. Протокол соединения создает виртуальные каналы, которые могут использоваться для различных целей. Существуют стандартные методы установки безопасных сессий интерактивного shell и перенаправления ("туннелирования") произвольных портов TCP/IP, соединений X11 и др.

Протокол транспортного подуровня ssh-trans протокола ssh.

Транспортный уровень является протоколом нижнего уровня и обеспечивает шифрование и целостность данных, а также аутентификацию сервера. Аутентификация пользователя данным протоколом не выполняется. Протокол используется для ведения переговоров о способе обмена ключа, алгоритме открытого ключа, алгоритме симметричного шифрования, алгоритме аутентификации сервера и хэш-алгоритме. Считается, что необходимы только две взаимные передачи для полного обмена ключами, аутентификации сервера, запроса и принятия уведомления о запросе. В худшем случае будет три взаимные передачи.

Установление соединения протоколом транспортного подуровня ssh-trans протокола ssh

Протокол ssh-trans работает поверх любого 8-битного надежного транспорта. Транспорт, расположенный ниже, должен защищать от ошибок передачи, т.к. при возникновении ошибки ssh-соединение прерывается.Соединение инициирует клиент.

Обмен версией протокола транспортного подуровня ssh-trans протокола ssh

Когда соединение уровня TCP установлено, обе стороны должны послать информационную строку в форме ssh-protoversion-softwareversion comments. Рассмотрим версию протокола 2.0. Обмен ключей начинается непосредственно после посылки данного сообщения. Все пакеты, следующие за строкой идентификации, будут использовать так называемый протокол бинарных пакетов.

Совместимость протокола транспортного подуровня ssh-trans со старыми версиями ssh

старый клиент и новая реализация сервера: имеется возможность поддержки конфигурационного флага совместимости со старыми версиями. Когда флаг установлен, сервер должен идентифицировать свой протокол версии как 1.99. Клиенты, использующие протокол 2.0, должны идентифицировать его как 2.0. В режиме совместимости сервер не должен посылать любые данные после своей строки идентификации до тех пор, пока не получит строку идентификации от клиента. После этого сервер может определить, использует ли клиент старый протокол, и в этом случае сам должен использовать старый протокол. Когда нет необходимости в совместимости со старыми клиентами, сервер может послать свои инициализационные данные обмена ключа непосредственно после строки идентификации.

новый клиент и старый сервер: новый клиент может немедленно послать дополнительные данные после своей строки идентификации, т.е. перед получением идентификации сервера. Поэтому старый протокол на сервере может исказить последовательность взаимодействия и клиент тем самым узнает, что сервер использует старый протокол. В этом случае клиент должен закрыть соединение с сервером и установить новое соединение, используя старый протокол.

Протокол аутентификации пользователя ssh-userauth протокола ssh

Протокол аутентификации пользователя выполняется поверх протокола ssh-trans. Данный протокол предполагает, что лежащий ниже протокол обеспечивает целостность и конфиденциальность. Идентификатор сессии данный протокол получает из протокола транспортного уровня. Протоколу аутентификации пользователя ssh-userauth необходима конфиденциальность, которая обеспечивается протоколом транспортного уровня. Аутентификацией клиента управляет сервер, посылая клиенту сообщение, что аутентификация может продолжаться. Клиент может выбирать в любом порядке один из перечисленных сервером методов. Это, с одной стороны, дает серверу возможность более комплексного управления аутентификационным процессом, а, с другой стороны, дает клиенту гибкость в использовании методов, которые он поддерживает или которые больше всего подходят для пользователя, если сервер предоставляет возможность аутентификации несколькими методами. Аутентификационные методы идентифицируются по имени. Метод none зарезервирован, но не должен быть в списке поддерживаемых. Сервер должен определить таймаут для аутентификации и разрывать соединение, если аутентификация не проведена за указанное время. Дополнительно должно быть установлено ограничение на число неудачных попыток аутентификации, которые клиент может предпринять в течение одной сессии. Если порог превышен, сервер должен разорвать соединение.

Запросы на аутентификацию пользователя ssh-userauth протокола ssh

Все запросы на аутентификацию должны использовать строго фиксированный формат сообщений. Определены только первые несколько полей; оставшиеся поля зависят от метода аутентификации.

```
SSH_MSG_USERAUTH_REQUEST
имя пользователя (UTF-8 кодировка)
имя сервиса (в US-ASCII)
имя метода (в US-ASCII)
остаток пакета определяется методом аутентификации.
```

Имя пользователя и сервис повторяются при каждой новой попытке аутентификации и могут изменяться. Сервер должен проверять это в каждом сообщении и полностью проверять любое добавленное аутентификационное состояние, если были сделаны подобные изменения. Имя сервиса определяет сервис, который будет запущен после завершения процесса аутентификации. Может быть определено несколько различных сервисов. Если запрашиваемый сервис недоступен, сервер может разорвать соединение немедленно или в любое более позднее время. Рекомендуется посылать соответствующее сообщение о разрыве соединения. В любом случае, если сервис не существует, аутентификация при этом не удачна. Если пользователь, который запрашивает аутентификацию, не существует, сервер может либо разорвать соединение, либо послать поддельный список приемлемых аутентификаций, но аутентификация при этом не удачна. Это дает возможность серверу избегать обнаружения информации о том, какие имена пользователей существуют. Аутентификационный запрос может привести к дальнейшему обмену сообщениями. Все такие сообщения зависят от используемого метода аутентификации, и клиент может в любое время продолжить обмен, послав новое сообщение `SSH_MSG_USERAUTH_REQUEST`. В этом случае сервер должен сбросить всю информацию о предыдущей попытке аутентификации и продолжить процесс аутентификации заново.

Ответы на запросы аутентификации пользователя ssh-userauth протокола ssh

Если сервер отклонил запрос на аутентификацию, он должен ответить следующим сообщением:

```
SSH_MSG_USERAUTH_FAILURE
аутентификации, которые могут быть продолжены
частичный успех
```

"Аутентификации, которые могут быть продолжены" – это разделенный запятыми список имен методов аутентификации, с использованием которых можно продолжать аутентификационный диалог. Считается, что серверы должны включать только те методы, которые реально

используются. Однако нигде не запрещено включать методы, которые не могут быть задействовать для аутентификации пользователя. Уже успешно завершенные аутентификации не должны включаться в список, если они по какой-либо причине не должны выполняться снова.
"Частичный успех" должен быть true, если запрос на аутентификацию, на который дается данный ответ, был успешным. Он должен быть false, если запрос не был успешно обработан.

Когда сервер принимает аутентификацию, он должен ответить:

```
SSH_MSG_USERAUTH_SUCCESS
```

Заметим, что это сообщение не посылается после каждого шага, если аутентификация осуществляется последовательно несколькими методами. Клиент может послать несколько аутентификационных запросов без ожидания ответов на предыдущие запросы. Сервер должен подтвердить все непрошедшие запросы сообщением

```
SSH_MSG_USERAUTH_FAILURE
```

Однако `SSH_MSG_USERAUTH_SUCCESS` должно посылаться только один раз, и после того, как оно послано, дальнейшие запросы на аутентификацию должны игнорироваться без каких-либо сообщений. На этом аутентификация завершается.

Сервер SSH может послать сообщение `SSH_MSG_USERAUTH_BANNER` в любое время перед успешным завершением аутентификации. Данное сообщение содержит текст, который будет показан пользователю. Формат сообщения следующий:

```
SSH_MSG_USERAUTH_BANNER
сообщение
language tag
```

По умолчанию клиент должен показать это сообщение на экране пользователя. Однако, поскольку сообщение, вероятно, будет посылаться при каждой попытке входа и клиентскому ПО придется открывать отдельное окно для этого напоминания, постольку клиентское ПО может разрешать пользователю полностью запрещать показ баннеров сервера. Если строка сообщения показывается, управляющие символы должны фильтроваться во избежание атак посылки управляющих терминальных символов.

Протокол соединения ssh-conn протокола ssh

Рассмотрим протокол соединения ssh-conn функционирующий поверх протокола ssh-auth. Он обеспечивает интерактивные входные сессии, удаленное выполнение команд, перенаправление TCP/IP-соединений и перенаправление X11-соединений, которые, при их наличии, объединяются в единственный зашифрованный туннель.

Общие запросы соединения ssh-conn протокола ssh

Существует несколько типов запросов, которые воздействуют на удаленный конец "глобально", независимо от канала. Примером является запрос на начало TCP/IP, перенаправляемого на определенный порт. Все подобные запросы используют следующий формат.

```
SSH_MSG_GLOBAL_REQUEST
имя запроса (ограничено US-ASCII)
ждать ответа
. . .   данные, специфичные для запроса
```

Получатель будет отвечать на данное сообщение `SSH_MSG_REQUEST_SUCCESS`, `SSH_MSG_REQUEST_FAILURE` или некоторым специфичным для запроса сообщением, если "ждать ответа" есть TRUE.

```
SSH_MSG_REQUEST_SUCCESS
```

Если получатель не распознает или не поддерживает запрос, он просто отвечает:

```
SSH_MSG_REQUEST_FAILURE
```

Механизм соединения ssh-conn протокола ssh

Все терминальные сессии, перенаправляемые соединения и т.п. весьма условно называемые каналами. Каждая из сторон может открыть канал. Несколько каналов

мультиплексируются в единственное соединение. Каналы идентифицируются номерами на каждом конце. Номера канала на разных концах могут различаться. Запросы на открытие канала содержат номер канала отправителя. Любые другие относящиеся к каналу сообщения содержат номер канала получателя. Каналы являются потокоуправляемыми. Никакие данные не могут быть посланы по каналу до тех пор, пока полученное сообщение не укажет, что пространство окна доступно. Пространство окна определяет количество данных, которое можно передать по каналу в данный момент времени.

Открытие соединения ssh-conn протокола ssh

Когда любая из сторон хочет открыть канал, она определяет локальный номер для канала и посылает другой стороне сообщение, включающее локальный номер канала и размер начального окна.

```
SSH_MSG_CHANNEL_OPEN
тип канала (ограничение US-ASCII)
канал отправителя
начальный размер окна
максимальный размер пакета
. . .   данные, специфичные для типа канала
```

Тип канала представляет собой имя, определенное в архитектуре протокола. "Канал отправителя" является локальным идентификатором канала, используемым отправителем данного сообщения. "Начальный размер окна" определяет, сколько байтов данных канала может быть послано отправителем данного сообщения без регулирования размера окна. "Максимальный размер пакета" определяет максимальный размер индивидуального пакета данных, который может быть послан отправителем. Например, можно использовать небольшие пакеты для интерактивных соединений для уменьшения времени ответа по медленным каналам. Затем удаленная сторона решает, может ли она открыть канал, и отвечает следующим сообщением

```
SSH_MSG_CHANNEL_OPEN_CONFIGURATION
канал получателя
канал отправителя
начальный размер окна
максимальный размер пакета
. . . данные, специфичные для типа канала
```

где "канал получателя" есть номер канала, указанный в первоначальном запросе открытия, и "канал отправителя" есть номер канала на другом конце, или

```
SSH_MSG_CHANNEL_OPEN_FAILURE
канал отправителя
код причины
дополнительная текстовая информация
    (ISO-10646 UTF-8)
тег языка
```

Если получатель `SSH_MSG_CHANNEL_OPEN` сообщения не поддерживает указанный тип канала, он отвечает `SSH_MSG_OPEN_FAILURE`.

Передача данных соединения ssh-conn протокола ssh

Размер окна определяет количество байтов, которое может быть послано другому участнику в случае необходимости регулирования размера окна.

```
SSH_MSG_CHANNEL_WINDOW_ADJUST
канал получателя
байты для добавления
```

После получения этого сообщения получатель может послать на указанное число байтов больше, чем было разрешено посылать раньше. Размер окна при этом возрастает. Данные передаются в сообщении следующего типа:

```
SSH_MSG_CHANNEL_DATA
канал получателя
данные
```

Максимально допустимое количество данных определяется текущим размером окна. Размер окна уменьшается на количество переданных данных. Оба участника могут игнорировать

все данные, посланные после того, как разрешенное окно стало пустым. Дополнительно некоторые каналы могут передавать несколько типов данных. Примером могут служить stderr данные из интерактивных сессий. Такие данные могут быть переданы в SSH_MSG_CHANNEL_EXTENDED сообщениях, где отдельное целое определяет тип данных. Допустимые типы и их интерпретация зависит от типа канала.

```
SSH_MSG_CHANNEL_EXTENDED_DATA
канал получателя
код типа данных
данные
```

Закрытие соединения ssh-conn протокола ssh

Когда участник более не собирается передавать данные по каналу, он должен послать:

```
SSH_MSG_CHANNEL_EOF
канал получателя
```

На данное сообщение явного ответа может не быть, однако приложение может послать EOF на другой конец канала. Заметим, что канал после такого сообщения остается открытым , и данные могут быть посланы в другом направлении. Данное сообщение не использует пространство окна и может быть послано даже в том случае, если доступного пространства нет. Когда один из участников хочет закрыть канал, он посылает SSH_MSG_CHANNEL_CLOSE. При получении данного сообщения участник должен послать обратно SSH_MSG_CHANNEL_CLOSE, если данное сообщение еще не было послано. Канал считается закрытым для участника, когда он и получил, и отправил SSH_MSG_CHANNEL_CLOSE, после чего участник может переиспользовать номер канала. Он может послать SSH_MSG_CHANNEL_CLOSE без отправления или получения SSH_MSG_CHANNEL_EOF.

```
SSH_MSG_CHANNEL_CLOSE
канал получателя
```

Данное сообщение не использует пространство окна и может быть послано даже в том случае, если доступного пространства нет. Рекомендуется, чтобы любые данные, посланные до этого сообщения, были по возможности доставлены реальному получателю.

Перенаправление порта TCP/IP соединения ssh-conn протокола ssh

Участнику нет необходимости посылать явный запрос перенаправления на другой порт. Однако, если он хочет иметь соединение с портом на другой стороне, перенаправленное на локальную сторону, он должен сделать явный запрос с помощью следующего сообщения:

```
SSH_MSG_GLOBAL_REQUEST
"tcpip-forward"
ждать ответа
адрес для связывания
номер порта для связывания
```

"Адрес для связывания" и "порт для связывания" определяют IP-адрес и порт, к которому будет привязан сокет. Можно фильтровать соединения на основе информации, полученной в запросе открытия. Перенаправление на привилегированные порты следует разрешать только в том случае, если пользователь аутентифицирован как привилегированный. Реализации клиента должны отвергать эти сообщения; обычно они посылаются только клиентом серверу. Порт перенаправления может быть отменен следующим сообщением. Запросы открытия канала могут быть получены до получения ответа на данное сообщение.

```
SSH_MSG_GLOBAL_REQUEST
"cancel-tcpip-forward"
ждать ответа
адрес для связывания
номер порта для связывания
```

Реализации клиента должны отвергать данное сообщение; обычно оно посылается только клиентом.

Достоинства протокола ssh:

Протокол ssh с использованием криптографической защиты передаваемой информации может защитить от:

"IP spoofing'a" (подмены ip-адреса), когда нарушитель (удаленное атакующее СВТ) высылает свои пакеты симулируя якобы они пришли с другого компьютера, с которого разрешен доступ. Ssh защищает от подмены даже в локальной сети, в случае, если нарушитель решил подменить маршрутизатор наружу "собой".

"IP source routing" (подмены ip-маршрутов), когда СВТ может имитировать IP-пакеты, которые как-бы приходят от другого, разрешенного компьютера

"DNS spoofing", когда нарушитель фальсифицирует записи "name server'a"

прослушивания нешифрованных паролей и других данных промежуточными компьютерами.

манипуляций над данными в промежуточных СВТ.

атак, основанных на прослушивании "X authentication data" и подлога соединения к X11 серверу.

Протокол ssh не доверяет среде передачи данных (ИКСС), т.е. нарушитель, имеющий определенный доступ к сети, может лишь разорвать установленное ssh соединение, но не расшифровать его (это зависит от имеющихся у нарушителя вычислительных мощностей и др. возможностей), похитить или переопределить передаваемые данные. Вместе с тем, протокол ssh имеет опцию шифрования "none", которая необходима исключительно для отладки и не должна использоваться для обычной работы.

Недостатки протокола ssh:

протокол ssh не защищает в случае нарушения безопасности СВТ какими-либо другими методами. Например, если нарушитель каким-либо иным методом получил привилегированный доступ к СВТ, то это позволит ему изменить настройки и работу протокола ssh.

если нарушитель получил доступ в домашний каталог пользователя, то безопасность протокола ssh так же под вопросом, поскольку в домашнем каталоге могут храниться соответствующие ключи протокола ssh (пример – доступ к домашнему каталогу по протоколу Network File System, NFS).

протокол ssh выполняется на верхнем уровне и предполагается, что аутентификация пользователя и защита против атак сетевого уровня обеспечивается протоколами нижнего уровня, при этом порты перенаправления могут потенциально допускать проникновение через внешние границы безопасности, такие как межсетевые экраны, поскольку данный протокол обычно выполняется внутри зашифрованного туннеля, межсетевые экраны не имеют возможности просматривать и анализировать трафик.

Использование протоколов telnet и ssh при решении задач ОТУ объективно целесообразно, поскольку фактически подавляющее большинство производителей ТКО реализовало в соответствующих версиях микропрограммного обеспечения ТКО так называемый cisco-like интерфейс командной строки, поэтому конфигурирование ТКО с использованием протоколов telnet и ssh стало более-менее единообразным. Иными словами, научившись конфигурировать какой либо один вид ТКО одного производителя инженер службы эксплуатации и/или программист службы развития достаточно быстро способен разобраться в синтаксисе команд управления ТКО другого производителя.

Безопасность использования протоколов telnet и ssh целесообразно реализовывать комплексно как имеющимися для протокола ssh встроенными возможностями, так и другими, например, выделением локальных сетей управления для соответствующих видов ТКО, например, с использованием средств виртуализации vlan на основе стандарта IEEE 802.1q. Рассмотрение этих решений выходит за рамки данной лекции.

Формат данных JSON.

JSON (JavaScript Object Notation) - это читаемый человеком формат данных, используемый web-приложениями и API для хранения, передачи и чтения данных. JSON достаточно просто воспринимать непосредственно и можно использовать с большинством

современных языков программирования, включая Python. JSON использует иерархическую структуру и содержит вложенные значения, использует фигурные скобки { } для хранения объектов и квадратные скобки [ ] для хранения массивов, данные записываются в виде пар ключ/значение.

В JSON данные, известные как объект, представляют собой одну или несколько пар ключ/значение, заключенных в фигурные скобки { }. Синтаксис для объекта JSON включает в себя:

ключи должны быть строками в двойных кавычках " ".

значения должны быть допустимым типом данных JSON (строка, число, массив, логический, нулевой или другой объект).

ключи и значения разделяются двоеточием.

несколько пар ключ/значение внутри объекта разделяются запятыми.

пробел не имеет значения.

Иногда ключ может содержать более одного значения. Это массив, который в JSON представляет собой упорядоченный список значений. Характеристики массивов в JSON включают в себя:

за ключом следует двоеточие и список значений в квадратных скобках [ ].

массив представляет собой упорядоченный список значений.

массив может содержать несколько типов значений, включая строку, число, логическое значение, объект или другой массив внутри массива.

каждое значение в массиве отделяется запятой.

Например, список адресов IPv4 может выглядеть следующим образом. Ключ - “addresses”. Каждый элемент в списке является отдельным объектом, разделенным фигурными скобками { }. Объектами являются две пары ключ/значение: адрес IPv4 («ip») и маска подсети («netmask»), разделенные запятой. Массив объектов в списке также разделяется запятой после закрывающей скобки для каждого объекта.

Пример вывода JSON для интерфейса Gigabit Ethernet:


```
{
  "ietf-interfaces:interface": {
    "name": "GigabitEthernet",
    "description": "Wide Area Network",
    "enabled": true,
    "ietf-ip:ipv4": {
      "address": [
        {
          "ip": "172.17.0.2",
          "netmask": "255.255.0.0"
        },
        {
          "ip": "172.17.0.3",
          "netmask": "255.255.0.0"
        },
        {
          "ip": "172.17.0.4",
          "netmask": "255.255.0.0"
        }
      ]
    }
  }
```

}

Формат данных YAML.

YAML (Yet Another Markup Language) - еще один вид формата данных, используемый приложениями для хранения, передачи и чтения данных, который похож на JSON и считается надмножеством JSON. YAML имеет минималистский формат, облегчающий чтение и запись и использует отступ для определения своей структуры, без использования скобок или запятых. Подобно JSON, объект YAML представляет собой одну или несколько пар ключ-значение. Пары ключ-значение отделяются двоеточием без использования кавычек. В YAML дефис используется для разделения каждого элемента в списке.

Пример вывода YAML для интерфейса Gigabit Ethernet:

```
ietf-interfaces:interface:
  name: GigabitEthernet
  description: Wide Area Network
  enabled: true
  ietf-ip:ipv4:
    address:
    - ip: 172.17.0.2
      netmask: 255.255.0.0
    - ip: 172.17.0.3
      netmask: 255.255.0.0
    - ip: 172.17.0.4
      netmask: 255.255.0.0
```

Заключение

На лекции были рассмотрены вопросы построения и взаимодействия клиент-серверного программного обеспечения без понимания которых не возможно создание качественного программного продукта. Повторение материала лекции с использованием рекомендованной литературы позволит хорошо закрепить теоретический материал. Использование реализаций клиент-серверного ПО для организации взаимодействия информационных систем и примеры будут рассмотрены на следующей лекции.