Internet Engineering Task Force (IETF)
Request for Comments: 5905
Obsoletes: 1305, 4330
Category: Standards Track
ISSN: 2070-1721

D. Mills
U. Delaware
J. Martin, Ed.
ISC
J. Burbank
W. Kasch
JHU/APL

June 2010

Network Time Protocol Version 4: Protocol and Algorithms Specification

Протокол сетевого времени NTPv4 - спецификация и алгоритмы

Аннотация

Протокол сетевого времени (Network Time Protocol или NTP) широко применяется для синхронизации клиентских часов в Internet. Этот документ описывает протокол NTP версии 4 (NTPv4), который совместим с NTP версии 3 (NTPv3, RFC 1305), а также с предыдущими версиями протокола. NTPv4 использует изменённый заголовок для работы с адресами семейства IPv6. В NTPv4 включены фундаментальные улучшения в алгоритмах смягчения и дисциплины, повышающие потенциальную точность до десятков микросекунд для современных рабочих станций и скоростных ЛВС. Протокол включает схему динамического обнаружения серверов, что во многих случаях позволяет избежать специальной настройки. В протоколе исправлены некоторые ошибки в устройстве и реализации NTPv3, а также включён дополнительный механизм расширения.

Статус документа

Документ относится к категории Internet Standards Track.

Документ является результатом работы IETF[1] и представляет согласованный взгляд сообщества IETF. Документ прошёл открытое обсуждение и был одобрен для публикации IESG[2]. Дополнительную информацию о стандартах Internet можно найти в разделе 2 в RFC 5741.

Информацию о текущем статусе документа, ошибках и способах обратной связи можно найти по ссылке http://www.rfc-editor.org/info/rfc5905.

Оглавление

[1]Internet Engineering Task Force - комиссия по решению инженерных задач Internet.
[2]Internet Engineering Steering Group - комиссия по инженерным разработкам Internet.

1. Введение

Этот документ определяет протокол NTP версии 4 (NTPv4), который широко применяется для синхронизации часов между множеством распределенных серверов и клиентов. Описана базовая архитектура, протокол, конечные автоматы, структуры данных и алгоритмы. NTPv4 расширяет функциональность NTPv3, описанную в [RFC1305], и Simple NTP версии 4 (SNTPv4), описанную в [RFC4330] (SNTPv4 является частью NTPv4). Этот документ отменяет [RFC1305] и [RFC4330]. Хотя в отдельные поля заголовков протокола внесены незначительные изменения, это не влияет на совместимость NTPv4 с предыдущими версиями NTP и SNTP.

Модель подсети NTP включает множество широкодоступных первичных серверов времени, синхронизируемых с национальными стандартами по проводным и радиоканалам. Назначение протокола NTP состоит в передаче сведений о точном времени от этих первичных серверов вторичным серверам и клиентам через частные сети и публичную сеть Internet. Точно настроенные алгоритмы снижают ошибки, которые могут возникать в результате сетевых нарушений, отказов серверов и возможных вредных воздействий. Серверы и клиенты настраиваются так, что значения передаются от первичных серверов в сторону клиентов через разветвлённую сеть вторичных серверов.

В NTPv4 устранены существенные недостатки NTPv3, исправлены некоторые ошибки и добавлены новые возможности. В частности, расширены определения меток времени NTP для использования типа данных с плавающей запятой (точкой) и двойной точностью. В результате разрешение (дискретность счета) времени составляет меньше 1 нсек, а частотное разрешение - меньше 1 нсек. За секунду. Дополнительные улучшения включают алгоритм дисциплины часов, который более стабилен при флуктуациях аппаратных часов системы. Типовые первичные серверы используют

современные машины и обеспечивают точность в пределах нескольких десятков мксек. Типовые вторичные серверы и клиенты в быстрых ЛВС обеспечивают точность в несколько сотен микросекунд с интервалами опроса до 1024 секунд, что было максимальным для NTPv3. С помощью NTPv4 клиенты и серверы работают с точностью в несколько десятков миллисекунд, а интервалы опроса составляют до 36 часов.

Основная часть документа описывает ядро протокола и структуры данных, требуемые для взаимодействия разных реализаций. В Приложении A приведён полнофункциональный пример в форме скелета программы, включающий структуры данных и сегменты кода для алгоритмов ядра, а также алгоритмов смягчения, служащих для повышения точности и надёжности. Хотя скелет программы и другие описания в этом документе относятся к конкретной реализации, они не являются единственным способом реализации требуемых функций. Содержимое Приложения A не является нормативным и предназначено для иллюстрации работы протокола, поэтому его не обязательно использовать в реализации. Хотя описанная в документе схема аутентификации NTPv3 с симметричным ключом была скопирована из NTPv3, новая схема аутентификации с открытым ключом Autokey доступна для NTPv4 [RFC5906].

NTP включает режимы работы, описанные в разделе 2, с использованием типов (раздел 6) и структур (раздел 7) данных. Модель реализации в разделе 5 описана на основе многопоточной, многопроцессорной архитектуры, хотя возможны и другие варианты. Протокол в линии (on-wire) описанный в разделе 8, основан на схеме с возвращаемым временем (returnable-time), которая зависит лишь от измеренных сдвигов (offset) часов и не требует надёжной доставки сообщений. Гарантированная доставка, такая как TCP [RFC0793], может сделать доставленный пакет NTP менее надёжным, поскольку повторы передачи увеличивают задержку и другие ошибки. Подсеть синхронизации является самоорганизующейся иерархической сетью «ведущий-ведомый» с путями синхронизации, определяемыми связующим деревом кратчайших путей (shortest-path spanning tree) и заданной метрикой. Хотя может существовать несколько ведущих серверов (primary server), протокол выбора среди них не требуется.

Этот документ включает материалы из [ref9], где содержатся схемы и формулы, не подходящие для формата RFC. Много дополнительных сведений представлено в [ref7], включая обширный технический анализ и оценку производительности протокола и алгоритмов из этого документа. Эталонная реализация доступна на www.ntp.org.

Далее в документе представлено множество численных переменных и математических выражений. Некоторые переменные обозначаются греческими символами, которые представлены полным именем буквы с учётом регистра. Например, DELTA указывает заглавную греческую букву «дельта», а delta - строчную. Кроме того, нижние индексы обозначены символом подчёркивания '_', например, theta_i указывает греческую букву theta с нижним индексом i (фонетически i). Время в документе указывается в секундах, частоты указаны значениями FFO (fractional frequency offset). Зачастую FFO удобно указывать в миллионных долях (ppm).

1.1. Уровни требований

Ключевые слова **должно** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **нужно** (SHALL), **не следует** (SHALL NOT), **следует** (SHOULD), **не нужно** (SHOULD NOT), **рекомендуется** (RECOMMENDED), **возможно** (MAY), **необязательно** (OPTIONAL) в данном документе интерпретируются в соответствии с [RFC2119].

2. Режимы работы

Реализация NTP работает как первичный сервер, вторичный сервер или клиент. Первичный сервер синхронизируется с эталонными часами, отслеживаемыми напрямую до UTC (например, GPS, Galileo и т. п.). Клиент синхронизируется с одним или несколькими серверами восходящего направления, но сам не обеспечивает синхронизации зависимым клиентам. Вторичный сервер имеет один или несколько восходящих серверов и один или несколько нисходящих серверов или клиентов. Все серверы и клиенты, полностью совместимые с NTPv4, **должны** реализовать весь набор алгоритмов, описанных в этом документе. Для поддержки стабильности в больших подсетях NTP вторичным серверам следует быть полностью совместимыми с NTPv4. **Можно** применять дополнительные алгоритмы, но их вывод **должен** быть идентичен выводу алгоритмов, описанных в этой спецификации.

3. Режимы протокола

Имеется 3 варианта протокола NTP: симметричный, клиент-сервер и широковещательный. С каждым из них связан режим ассоциации (описание отношения между двумя узлами NTP) как показано на рисунке 1. Кроме того, постоянные ассоциации активируются (mobilize) при старте и никогда не деактивируются (demobilize), а временные (эфемерные — ephemeral) активируются приёмом пакета и деактивируются по ошибке или тайм-ауту.

Режим ассоциации	*Значение режима ассоциации*	*Значение пакетного режима*
Симметричный активный	1	1 или 2
Симметричный пассивный	2	1
Клиент	3	4
Сервер	4	3
Широковещательный сервер	5	-
Широковещательный клиент	6	5

Рисунок 1. Режимы протокола

В варианте клиент-сервер постоянный клиент передаёт пакет пакетного режима 3 серверу, а тот отвечает пакетами пакетного режима 4[1]. Серверы обеспечивают синхронизацию одного или нескольких клиентов, но не принимают синхронизацию от них. Сервер может также быть драйвером эталонных часов, получая время напрямую от стандартного источника, такого как приёмник GPS или телефонная модемная служба. В этом варианте клиент вытягивает синхронизацию у серверов.

В симметричном варианте партнёры выступают сразу клиентами и серверами используя активную или пассивную симметричную ассоциацию. Постоянная активная ассоциация передаёт симметричные активные пакеты (режим 1) активной симметричной ассоциации партнёра. Временная симметричная ассоциация может быть активизирована получением симметричного активного пакета без соответствующей ассоциации. Эта ассоциация передаёт симметричные пассивные пакеты (режим 2) и сохраняется до ошибки или тайм-аута. Партнёры выталкивают и

[1]В оригинале это предложение и 2 последних строки таблицы содержат ошибки. См. https://www.rfc-editor.org/errata/eid3126. *Прим. перев.*

втягивают синхронизацию друг от друга. В этом документе партнёр рассматривается как клиент, поэтому упонимание клиента может относиться и к партнёру.

В широковещательном варианте ассоциация постоянного широковещательного сервера передаёт периодические пакеты широковещательного сервера (режим 5), которые может получить множество клиентов. При получении такого пакета без соответствующей ассоциации ассоциация временного широковещательного клиента (режим 6) активизируется и сохраняется до ошибки или тайм-аута. Полезно обеспечить начальный поток (volley), где клиент, работающий в клиентском режиме обменивается несколькими пакетами с сервером, чтобы откалибровать задержку распространения и запустить протокол защиты Autokey, после чего клиент возвращаетмя в режим широковещательного клиента. Широковещательный сервер выталкивает синхронизацию клиентам и другим серверам.

Следуя в общих чертах принятым в телефонной индустрии соглашениям, уровень каждого сервера в иерархии определяется номером слоя (stratum). Первичным серверам назначается слой 1, вторичным серверам на каждом уровне назначается слой на 1 больше, чем в вышележащего уровня. По мере роста номера слоя точность снижается в зависимости от конкретного сетевого пути и стабильности системных часов. Средние ошибки, измеряемые дальнстью синхронизации, растут пропорционально номеру слоя и измеренной задержке кругового обхода (round-trip delay).

В качестве стандартной практики топологию сети синхронизации следует организовывать так, чтобы не возникало петель синхронизации и минимизировались дистанции синхронизации. В NTP топология подсети определяется с использованием варианта алгоритма распределенной маршрутизации Беллмана-Форда (Bellman-Ford), который рассчитывает связующее дерево кратчайших путей с первичным сервером в качестве корня. В результате алгоритм автоматически реорганизует подсеть, чтобы обеспечить наиболее точное и надёжное время даже при сбоях в сети синхронизации.

3.1. Динамическое обнаружение серверов

Две специальные ассоциации - manycast-клиент и manycast-сервер, - которые обеспечивают функцию динамического обнаружения сервера. Имеется два типа клиентских ассоциаций manycast: постоянные и временные. Постоянный manycast-клиент передаёт клиентские пакеты (режим 3) по назначенному групповому или широковещательному адресу IPv4 или IPv6. Назначенные manycast-серверы в диапазоне срока действия пакета (time-to-live или TTL) в заголовке прослушивают пакеты по этому адресу. Если сервер подходит для синхронизации, он возвращает обычный серверный пакет (режим 4) по индивидуальному адресу клиента. При получении этого пакета клиент активирует временную клиентскую ассоциацию (режим 3), которая сохраняется до ошибки или тайм-аута.

Клиент manycast продолжает передавать пакеты поиска для создания минимального числа ассоциаций. Передача начинается с TTL=1 с увеличением значения, пока не будет создано минимальное число ассоциаций или достигнуто максимальное значение TTL. Если по достижении максимума TTL активировано ещё недостаточно ассоциаций, клиент останавливает передачу на время ожидания для очистки всех ассоциаций, затем повторяет цикл поиска. Если минимальное число ассоциаций активировано, клиент начинает передавать один пакет за время ожидания для поддержки ассоциаций. Значение поля ограничено диапазоном от 1 до 255, но приложения могут менять диапазон.

Временная ассоциации конкурируею мжду собой. При активизации новой временной ассоциации клиент запускает алгоритмы смягчения, описанные в разделе 10 и параграфе 11.2, для выбора наиболее подходящих ассоциаций из числа временных. Остальные временные ассоциации завершаются по тайм-ауту. Таким образом, популяция включает только лучших кандидатов из числа недавно ответивших пакетом NTP, для дисциплины системных часов.

4. Определения

В этом разделе определено множество технических терминов. Шкала времени (timescale) - это система отсчёта, где время выражается монотонно возрастающим двоичным счётчиком с неопределённым числом битов. Счет ведётся в секундах и долях секунды, если используется десятичная точка (запятая). Шкала универсального координированного времени (Coordinated Universal Time или UTC) определена в ITU-R TF.460 [ITU-R_TF.460]. Под эгидой метрической конвенции (Metre Convention) 1865 г. в 1975 году Генеральная конференция мер и весов (CGPM) [CGPM] настоятельно рекомендовала использовать UTC в качестве базы гражданского времени.

Шкала UTC представляет среднее солнечное время, распространяемое национальными органами стандартизации. Системное время представляется часами системы, поддерживаемыми оборудованием и операционной системой. Целью алгоритмов NTP является минимизация разности во времени и частоте (скорости хода) между UTC и системными часами. Когда эта разница ниже номинальных допусков (nominal tolerance), системные часы считаются синхронизированными с UTC.

Датой события считается время UTC, при котором событие произошло. Даты - это эфемерные значения, обозначаемые символом T. Время при работе (running time) - это шкала времени, совпадающая с функцией синхронизации программы NTP.

Метка времени T(t) представляет дату UTC или смещение от UTC в момент t (running time). Какой из двух смыслов применяется должно быть ясно из контекста. Пусть T(t) - смещение по времени (time offset), R(t) - смещение по частоте (frequency offset), D(t) - скорость старения (первая производная R(t) по t). Тогда, если T(t_0) - смещение по времени UTC, определённое в момент t = t_0, смещение по времени UTC в момент t составляет

```
T(t) = T(t_0) + R(t_0)(t-t_0) + 1/2 * D(t_0)(t-t_0)^2 + e
```

где e - стохастическая ошибка (см. ниже). Хотя значение D(t) важно при характеризации точности задающих генераторов, этим значением обычно пренебрегают для компьютерных тактовых генераторов. В этом документе все значения времени даются в секундах, а значения допуска частоты в секундах за секунду (s/s). Иногда удобно выражать частотные сдвиги в миллионных долях 10^{-6}.

В компьютерных хронометрических приложениях важно оценивать производительность хронометрической функции. Модель производительности NTP включает 4 статистических показателя, обновляемые при каждом измерении (корректировке) времени с сервером. Смещение (theta) представляет максимально вероятное смещение часов сервера относительно системных часов. Задержка (delta) представляет круговую (round-trip) задержку между клиентом и сервером. Дисперсия (epsilon) указывает максимальную ошибку, присущую измерению, которая растёт со скоростью, равной максимально допустимому допуску тактовой частоты в дисциплине системных часов (PHI), который обычно

составляет 15 ppm. Вариации (jitter, psi) определяются как среднеквадратичное (root-mean-square или RMS) значение последних разностей смещений и указывает номинальную ошибку при оценке смещения.

Параметры theta, delta, epsilon, psi представляют измерения системных часов раздельно по отношению к каждому серверу, а протокол NTP включает механизмы объединения статистики для нескольких серверов, улучшающие дисциплину и точность калибровки системных часов. Смещение системы (THETA) представляет максимально вероятное смещение часов относительно совокупности серверов. Вариации (PSI) представляют номинальную ошибку при оценке смещения системы. Статистика delta и epsilon аккумулируется от эталонных часов для каждого слоя и дают корневую задержку (DELTA) и дисперсию (EPSILON). Дистанция синхронизации (LAMBDA), равная EPSILON + DELTA / 2, указывает максимальную ошибку для всех случаев. Подробное описание этих показателей статистики приведено в параграфе 11.2. Операции системного процесса. Параметры доступны зависимым приложениям для оценки производительности функции синхронизации.

5. Модель реализации

На рисунке 2 показана архитектура типовой многопотоковой реализации. Она включает 2 выделенных процесса на каждый сервер - партнёрский (peer) процесс для получения сообщений от сервера или эталонных часов и процесс опроса (poll) для передачи сообщений серверу или эталонным часам.

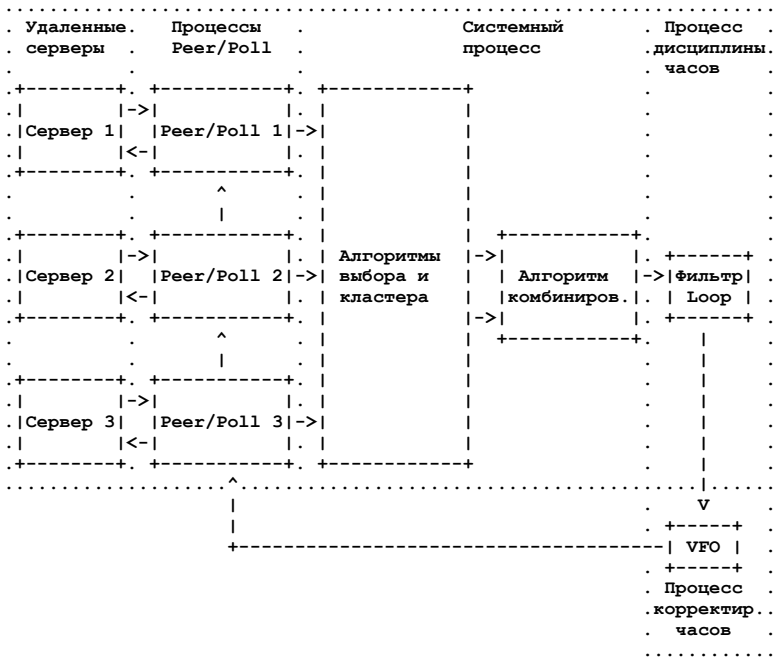

Рисунок 2. Модель реализации.

Эти процессы работают с общей структурой данных, называемой ассоциацией, которая содержит описанные выше показатели статистики вместе с различными структурами данных, описанными в разделе 9. Клиент передаёт пакеты одному или нескольким серверам, а затем обрабатывает возвращённые пакеты. Сервер меняет местами адреса и номера портов отправителя и получателя, переписывает некоторые поля в пакете и возвращает пакет сразу (в режиме клиент-сервер) или спустя некоторое время (в симметричном режиме). При получении каждого сообщения NTP рассчитывается смещение theta между часами системы и партнёра, а также связанная с статистика delta, epsilon, psi.

Системный процесс включает алгоритмы выбора, кластеризации и комбинирования, смягчающие различия между серверами и эталонными часами для определения наиболее точных и надёжных кандидатов для синхронизации системных часов. Алгоритм выбора использует византийские (Byzantine) принципы обнаружения отказов для отбрасывания предположительно неподходящих кандидатов, называемых «фальшивыми часами» (falsetickers), оставляя лишь хороших кандидатов, называемых «истинными часами» (truechimers). Истинными считаются часы, которые обеспечивают точность хронометража в соответствии с ранее опубликованным и доверенным стандартом, а фальшивыми - часы, которые показывают неверное или несогласованное время. Алгоритм кластеризации использует принципы статистики для выбора наиболее подходящего набора истинных часов. Алгоритм комбинирования рассчитывает итоговое смещение часов путём статистического усреднения оставшихся истинных часов.

Процесс дисциплины часов является системным процессом, контролирующим время и частоту системных часов, представленные генератором переменной частоты (variable frequency oscillator или VFO). Временные метки (тики) от VFO замыкают контур обратной связи, поддерживающей системное время. С процессом дисциплины связан процесс корректировки часов (clock-adjust), запускаемый 1 раз в секунду для внесения расчётного сдвига часов и поддержки постоянной частоты. Значение RMS разностей прошлых смещений времени представляет номинальную ошибку или вариации (jitter) системных часов. RMS разностей прошлых смещений частоты представляет стабильность тактового генератора или дрейф частоты. Точная интерпретация этих терминов представлена в параграфе 11.3.

Клиент передаёт сообщения каждому серверу с интервалом 2^{tau} секунд, определяемому показателем опроса tau. В NTPv4 параметр tau составляет от 4 (16 секунд) до 17 (36 часов). Значение tau определяет алгоритм дисциплины часов, чтобы соответствовать постоянной цикла времени $T_c = 2^{tau}$. В режиме клиент-сервер серверы отвечают незамедлительно, однако в симметричных режимах каждый из двух партнёров поддерживает tau как функцию текущего смещения и вариаций системы, поэтому партнёры могут не согласовать общее значение. Важно, чтобы динамическое поведение алгоритма дисциплины часов тщательно контролировалось для поддержки стабильности подсети NTP в целом. Это требует от партнёров согласования общего значения tau, равного меньшему из двух показателей опроса среди партнёров. Протокол NTP включает средства надлежащего согласования этого параметра.

Модель реализации включает некоторые средства установки и корректировки системных часов. Предполагается, что операционная система предоставляет две функции - одну для установки времени напрямую (например, функция Unix

settimeofday()), другую для корректировки времени на небольшое значение вперёд или назад (например, функция Unix adjtime()). Здесь и далее скобки после имени отличают функцию от простой переменной. В предлагаемом решении процесс дисциплины часов использует adjtime(), если корректировка меньше заданного порога, и settimeofday() - в иных случаях. Способ корректировки и значение порога рассматриваются в разделе 10. Алгоритм фильтрации часов.

6. Типы данных

Все значения времени в NTP представляются в формате дополнение до 2 с порядком битов big-endian (см. Приложение А к [RFC0791]), нумеруемых с 0 для старшего (левого) бита. Имеется 3 формата времени NTP, 128-битовые даны, 64-битовые метки и 32-битовые короткие метки, как показано на рисунке 3. 128-битовые даты применяются в тех случаях, когда имеется достаточно места. Значение включает 64-битовое целое число со знаком для указания целых секунд, что позволяет представить 584 миллиарда лет, и 64-битовое значение дробной части с дискретностью 0,05 аттосек ($0{,}05*10^{-18}$[1]). Для удобства сопоставления форматов поле seconds поделено на два поля - 32-битовое значение Era Number и 32-битовое значение Era Offset. Эры не могут создаваться NTP напрямую, да это и не требуется. При необходимости это можно сделать внешними средствами, такими как файловая система или специальное оборудование.

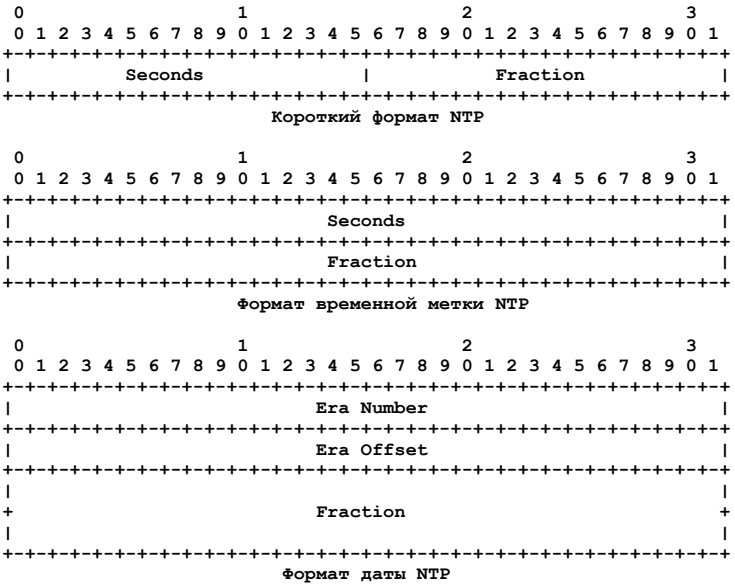

Рисунок 3. Форматы времени NTP.

64-битовые метки времени применяются в заголовках пакетов и других местах с ограниченным пространством. Они включают 32-битовое целое число без знака для указания секунд, что охватывает интервал в 136 лет, и 32-поле долей секунд с разрешением 232 пикосекунды. 32-битовый короткий формат применяется полях заголовков задержки и дисперсии, где полное разрешение и диапазон других форматов не оправданы. Это формат включает 16-битовое целое число без знака для учёта секунд и 16-битовое поле долей секунд. В форматах даты и временных меток первичной эпохой или базовой датой служит Era 0, начинающаяся в полночь (0 часов) 1 января 1900 г. по часовому поясу UTC, когда все биты имеют значение 0. Следует отметить, что, строго говоря, UTC не существовало до 1 января 1972 г., но удобно предполагать что UTC было всегда, даже если все исторические сведения о високосных секундах были утеряны. Даты исчисляются относительно этой первичной эпохи и отрицательные значения представляют время до начала эпохи. Отметим, что поле Era Offset в формате даты и поле Seconds в формате временных меток интерпретируются одинаково.

Временные метки являются беззнаковыми и операции над ними дают результат из той же или смежной эры. Эра 0 включает даты от начала первичной эпохи до некого момента в 2036 г., когда поле временной метки достигнет максимума и будет создана базовая дата эры 1. В любом формате метка со значением 0 указывает неизвестное или несинхронизированное время. На рисунке 4 приведены знаменательные даты NTP с соответствующими изменёнными юлианскими датами (Modified Julian Day или MJD), эрой и меткой NTP.

Дата	*MJD*	*Эра NTP*	*Смещение временной метки эры NTP*	*Эпоха*
1 января -4712 г.	-2400001	-49	1795583104	1-й день Юлианского календаря
1 января -1 г.	-679306	-14	139775744	2 BCE
1 января 0 г.	-678941[2]	-14	171311744	1 BCE
1 января 1 г.	-678575	-14	202934144[3]	1 CE
4 октября 1582 г.	-100851	-3	2873647488	Последний день Юлианского календаря
15 октября 1582 г.	-100840	-3	2874597888	Первый день Григорианского календаря
31 декабря 1899 г.	15019	-1	4294880896	Последний день NTP Era -1
1 января 1900 г.	15020	0	0	Первый день NTP Era 0
1 января 1970 г.	40587	0	2208988800	Первый день UNIX
1 января 1972 г.	41317	0	2272060800	Первый день современной UTC[4]
31 декабря 2000[5] г.	51909	0	3187209600	Последний день XX века
8 февраля 2036 г.	64731	1	63104	Первый день NTP Era 1

Рисунок 4. Интересные исторические даты NTP.

[1] В оригинале ошибочно указано 0.5e-18. См. https://www.rfc-editor.org/errata/eid2826. *Прим. перев.*
[2] В оригинале ошибочно указано -678491. См. https://www.rfc-editor.org/errata/eid4025. *Прим. перев.*
[3] В оригинале ошибочно указано 202939144. См. https://www.rfc-editor.org/errata/eid6524. *Прим. перев.*
[4] В оригинале слово «современной» отсутствовало. См. https://www.rfc-editor.org/errata/eid6423. *Прим. перев.*
[5] В оригинале ошибочно указан 1999 год и числа строки привязаны к нему. См. https://www.rfc-editor.org/errata/eid5189. *Прим. перев.*

Если p - число значимых битов во второй части, тогда разрешение часов в секундах будет определяться значением 2^{-p}. Чтобы минимизировать смещение и сделать метки времени непредсказуемыми для нарушителя, для незначащих битов следует устанавливать случайную битовую строку без смещения. Точность часов определяется как время на считывание часов. Отметим, что определённая таким образом точность может быть больше или меньше разрешения (дискретности) часов. Элемент rho, представляющий точность в этом протоколе, является большим из этих двух.

Единственной разрешённой для дат и меток времени арифметической операцией является вычитание с дополнением до 2, дающее 126- или 64-битовое значение со знаком. Крайне важно, чтобы в разности первого порядка между двумя датами сохранялось полное 128-битовое значение, а в разности меток - полное 64-битовое. Однако разность обычно мала по сравнению с интервалом секунд, поэтому её можно преобразовать в действительное число с плавающей запятой для дальнейшей обработки без потери точности. Важно отметить, что в арифметике с дополнением до 2 нет различий между значениями со знаком и без него (хотя при сравнении знак учитывается), различия проявляются только при ветвлении. Таким образом, несмотря различие дат со знаком и беззнаковых меток, они обрабатываются одинаково. Опасность при операциях с 64-битовыми метками времени, охватывающими эпоху, может возникать, например, в 2036 г., приводя к переполнению. На деле, если клиент был установлен в интервале 68 лет от сервера до запуска протокола, корректные значения будут получены даже при нахождении клиента и сервера в смежных эрах.

Некоторые значения времени, включая точность, временные константы и интервал опроса, представляются в экспоненциальном формате. Они представляются 8-битовыми целыми числами со знаком для двоичного логарифма от числа секунд. Для таких значений разрешены лишь операции инкремента и декремента. В настоящем документе для упрощения представления указание на такие переменные имени означает значение показателя, например, для интервала опроса в 1024 секунды имя и показатель будут давать фактическое значение (в данном случае показатель составляет 10).

Для преобразования системного времени в любом формате в дату или временную метку NTP требуется определить число секунд от начала первичной эпохи до системного времени. Целочисленный номер эры и метка времени определяются как,

```
era = s / 2^(32)
timestamp = s - era * 2^(32)
```

Эти выражения применимы для положительных и отрицательных дат. Для получения метки времени из значений the era и timestamp служит выражение

```
s = era * 2^(32) + timestamp.
```

Преобразование между NTP и системным временем может быть немного запутанным и выходит за рамки этого документа. Отметим, что число дней эры 0 на 1 больше числа дней во многих других эрах и это повторится уже около 2400 г. в эру 3.

В последующих описаниях переменных состояния явное указание целочисленного типа (integer) предполагает 32-битовое целое число без знака. Это урощает проверку границ, поскольку требуется задать лишь верхний предел. Без явного указания предполагается 64-битовое действительное число с плавающей запятой и двойной точностью. Исключения указываются, когда это необходимо.

7. Структуры данных

В последующих параграфах заданы конечные автоматы NTP. Переменные состояния разделены на классы по их функции в заголовках пакетов, процессах опроса и партнёрских процессах, системных процессах и процессах дисциплины часов. Переменные пакетов представляют в NTP значения заголовков в передаваемых и принимаемых пакетах. Переменные партнёра и опросов представляют содержимое ассоциации отдельно для каждого сервера. Системные переменные указывают состояние сервера с точки зрения зависимых от него клиентов. Переменные дисциплины часов показывают внутреннюю работу алгоритма дициплины. Примеры приведены в Приложении A.

7.1. Соглашения о структурах

Чтобы различать одноименные переменные из разных процессов применяются префиксы, показанные на рисунке 5. Переменная принимаемого пакета v, являющаяся членом структуры пакета r, имеет полное обозначение r.v. Аналогично, x.v будет переменной передаваемого пакета, p.v - переменной партнёра, s.v - системной переменной, c.v - переменной дисциплины часов. Имеется набор партнёрских переменных для каждой ассоциации и лишь по одному набору системных переменных и переменных часов.

Имя	*Описание*
r.	Переменная заголовка принимаемого пакета
x.	Переменная заголовка передаваемого пакета
p.	Переменная партнёра/опроса
s.	Системная переменная
c.	Переменная дисциплины часов

Рисунок 5. Соглашения о префиксах.

7.2. Глобальные параметры

В дополнение к классам переменных документ определяет глобальные переменные, включая показанные на рисунке 6.

Указаны лишь глобальные переменные требуемые для совместимости и любой реализации потребуется больший набор. В Приложении A.1.1 показаны глобальные переменные используемые в скелете кода для алгоритмов смягчения, дисциплины часов и связанных с ними функций, зависящих от реализации. Некоторые из значений параметров неизменны (например, номер порта NTP, выделенный IANA, и номер версии протокола NTPv4). Другие, например допуск по частоте (PHI), включают допущение о наихудшем поведении системных часов, синхронизированных однократно с последующим дрейфом, когда источник синхронизации становится недоступным. Минимальные и мксимальны значения параметров задают пределы переменных состояния в последующих параграфах документа.

Имя	*Значение*	*Описание*

PORT	123	NTP port number
VERSION	4	NTP version number
TOLERANCE	15*10-6	frequency tolerance PHI (s/s)
MINPOLL	4	minimum poll exponent (16 s)
MAXPOLL	17	maximum poll exponent (36 h)
MAXDISP	16	maximum dispersion (16 s)
MINDISP	0,005	minimum dispersion increment (s)
MAXDIST	1	distance threshold (1 s)
MAXSTRAT	16	maximum stratum number

Рисунок 6. Глобальные параметры.

Хотя переменные в этом документе показаны с фиксированными значениями, некоторые реализации могут делать их настраиваемыми в конфигурации. Например, эталонная вычисляет значение PRECISION как двоичный логарифм (log2) минимального времени среди нескольких итераций считывания системных часов.

7.3. Переменные заголовков пакетов

Наиболее важны с внешней точки зрения переменные состояния представлены на рисунке 7 и далее. Заголовок пакета NTP из целого числа 32-битовых (4 октета) слов с сетевым порядком байтов. Пакет включает заголовок, а также может включать одно или несколько полей расширения и код проверки подлинности (message authentication code или MAC). Компоненты заголовка идентичны применяемым в NTPv3 и предыдущих версиях. Необязательные поля расширения применяются криптографическими алгоритмами открытых ключей Autokey, как описано в [RFC5906]. Необязательное поле MAC используется Autokey и криптографическими алгоритмами, описанными в этом RFC.

Имя	***Формула***	***Описание***
leap	leap	Индикатор високосных секунд (LI)
version	version	Номер версии (VN)
mode	mode	Режим
stratum	stratum	Слой (stratum)
poll	poll	Показатель опроса (poll exponent)
precision	rho	Показатель точности)
rootdelay	delta_r	Корневая задержка (root delay)
rootdisp	epsilon_r	Корневая дисперсия (root dispersion)
refid	refid	Идентификатор эталона (reference ID)
reftime	reftime	Временная метка эталона
org	T1	Временная метка источника (origin)
rec	T2	Временная метка приёма
xmt	T3	Временная метка передачи
dst	T4	Временная метка получателя
keyid	keyid	Идентификатор ключа
dgst	dgst	Дайджест (подпись) сообщения

Рисунок 7. Переменные заголовков пакетов.

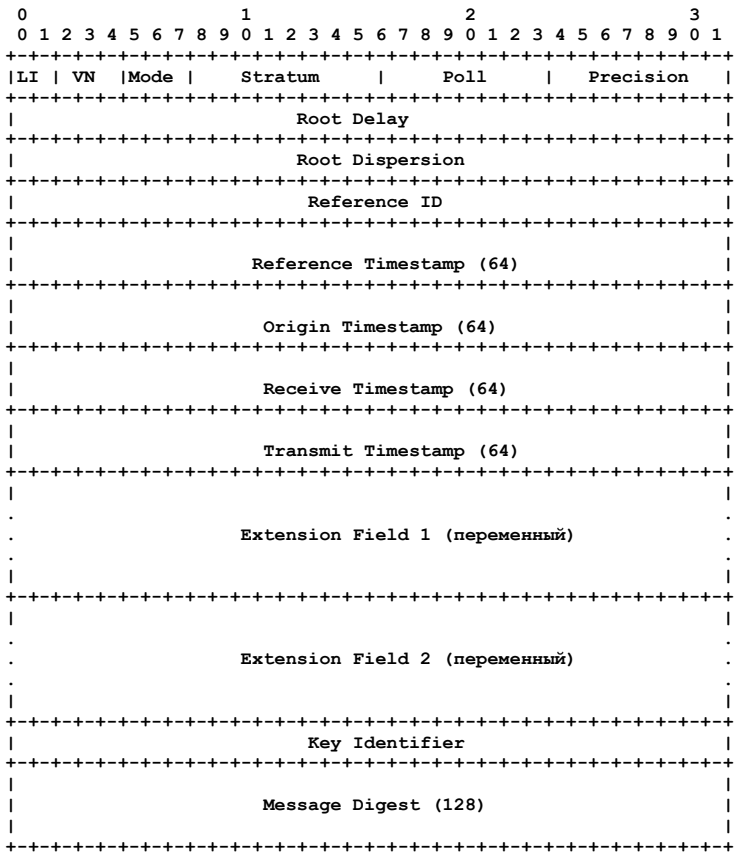

Рисунок 8. Формат заголовка пакета.

Пакет NTP передаётся в дейтаграмме UDP [RFC0768]. Некоторые поля используют несколько слов, а иные размещаются в меньших полях внутри слова. Заголовок пакета NTP, показанный на рисунке 8, имеет 12 слов, за

которыми могут следовать поля расширения и код аутентификации (MAC), состоящий из полей идентификатора ключа и дайджеста сообщения.

Поля расширения служат для добавления необязательных возможностей, например, протокола безопасности Autokey [RFC5906]. Формат поля расширения предназначен для обеспечения возможности его разбора без знания функций поля. Код MAC применяется в Autokey и схеме симметричного шифрования.

Переменные заголовков пакетов представлены на рисунке 7 и подробно описаны ниже. За исключением небольших изменений, связанных с использованием адресов семейства IPv6, эти поля совместимы с NTPv3. Поля заголовков пакетов применяются для передаваемых (префикс x) и примаемых (префикс r) пакетов. На рисунке 8 размер нескольких многословных полей указан в битах, если он отличается от принятых по умолчанию 32 битов. Базовый заголовок простирается от начала пакета до конца поля Transmit Timestamp. Поля и связанные с ними переменные (в скобках) описаны ниже.

LI Leap Indicator (leap)

2-битовое целочисленное предупреждение о приближающейся високосной секунде, добавляемой или удаляемой в последнюю минуту суток[1], как указано на рисунке 9.

Значение	*Смысл*
0	Нет предупреждений
1	Последняя минута суток содержит 61 секунду
2	Последняя минута суток содержит 59 секунд
3	Неизвестно (часы не синхронизированы)

Рисунок 9. Индикатор високосной секунды.

VN Version Number (version)

3-битовое целое число, представляющее номер версии NTP. Текущее значение - 4.

Mode (mode)

3-битовое целое число, представляющее режим. Значения режима приведены на рисунке 10.

Значение	*Смысл*
0	Резерв
1	Симмерична активная
2	Симмерична пассивная
3	Клиент
4	Сервер
5	Широковещательная
6	Управляющее сообщение NTP
7	Резерв для приватного использования

Рисунок 10. Режимы ассоциаций.

Stratum (stratum)

8-битовое целое число, представляющее слой (Рисунок 11).

Значение	*Смысл*
0	Не задан или недействителен
1	Первичный сервер (например, оборудованный приемником GPS)
2-15	Вторичный сервер (через NTP)
16	Не синхронизирован
17-255	Резерв

Рисунок 11. Слой пакета.

Обычно значение 0 в полученных пакетах сопоставляется со значением MAXSTRAT (16) в переменной партнёра p.stratum, а значения p.stratum = MAXSTRAT и выше сопоставляются с 0 в передаваемых пакетах. Это позволяет удобно умещать эталонные часы, которые обычно относятся к слою 0, с использованием того же лагоритма выбора чассов, который применяется для внешних источников (см. для примера Приложение A.5.5.1).

Poll

8-битовое целое число, представляющее максимальный интервал между последовательными сообщениями двоисным логарифмом числа секунд. По умолчанию для минимального и максимального интервалов опроса предложены значения 6 и 10, соответственно.

Precision

8-битовое целое число, представляющее точность системных часов двоичным логарифмом числа секнд. Например, значение -20[2] соответствует точности около 1 мксек. Точность может определяться при первом запуске службы как минимальное время нескольких операций считывания системных часов.

Root Delay (rootdelay)

Суммарная круговая задержка для эталонных часов в коротком формате NTP.

Root Dispersion (rootdisp)

Суммарная дисперсия для эталонных часов в коротком формате NTP.

Reference ID (refid)

32-битовый код, указывающий конкретный сервер или эталонные часы. Интерпретация значения зависит от поля stratum. При stratum=0 (не задано или недействительно) это 4-символьная строка ASCII [RFC1345], называемая kiss-кодом и служащая для отладки и мониторинга. При stratum=1 (эталонные часы) это 4-октетная, выровненная слева и дополненная справа нулями строка ASCII, назначенная эталонным часам. Полномочный список идентификаторов эталонных часов поддерживается IANA (Рисунок 12), а строки, начинающиеся символом ASCII X зарезервированы для нерегистрируемых значений, применяемых в экспериментах и разработке.

ID	*Источник времени*
GOES	Geosynchronous Orbit Environment Satellite
GPS	Global Position System
GAL	Galileo Positioning System
PPS	Generic pulse-per-second

[1] В оригинале ошибочно сказано «месяца». См. https://www.rfc-editor.org/errata/eid4504. *Прим. перев.*

[2] В оригинале ошибочно сказано -18. См. https://www.rfc-editor.org/errata/eid2476. *Прим. перев.*

IRIG	Inter-Range Instrumentation Group
WWVB	LF Radio WWVB Ft. Collins, CO 60 кГц
DCF	LF Radio DCF77 Mainflingen, DE 77.5 кГц
HBG	LF Radio HBG Prangins, HB 75 кГц
MSF	LF Radio MSF Anthorn, UK 60 кГц
JJY	LF Radio JJY Fukushima, JP 40 кГц, Saga, JP 60 кГц
LORC	MF Radio LORAN C station, 100 кГц
TDF	MF Radio Allouis, FR 162 кГц
CHU	HF Radio CHU Ottawa, Ontario
WWV	HF Radio WWV Ft. Collins, CO
WWVH	HF Radio WWVH Kauai, HI
NIST	Телефонный модем NIST
ACTS	Телефонный модем NIST
USNO	Телефонный модем USNO
PTB	Европейский телефонный модем

Рисунок 12. Идентификаторы источников точного времени.

Для stratum > 1 (вторичные серверы и клиенты) указывает сервер и может служить для обнаружения петель синхронизации. При использовании адресов семейства IPv4 идентификатором служит 4-октетный адрес IPv4, при использовании IPv6 - первые 4 октета хэш-значения MD5 для адреса IPv6. Отметим, что при использовании семейства IPv6 на сервере NTPv4 с клиентом NTPv3 поле Reference Identifier будет случайным значением и не сможет применяться для обнаружения петель синхронизации.

Reference Timestamp

Время, когда системные часы последний раз устанавливались или корректировались (метка NTP).

Origin Timestamp (org)

Время клиента в момент отправки запроса серверу (метка NTP).

Receive Timestamp (rec)

Время сервера в момент прибытия запроса от клиента (метка NTP).

Transmit Timestamp (xmt)

Время сервера в момент отправки отклика клиенту (метка NTP).

Destination Timestamp (dst)

Время клиента в момент прибытия отклика от сервера (метка NTP).

Примечание. Поле Destination Timestamp не включается как поле заголовка, оно определяется по прибытии пакета и становится доступным в структуре данных буфера пакетов.

Если NTP имеет доступ к физическому уровню, временные метки связываются с началом символа после старта кадра. В иных случаях реализации следует пытаться связать метку с наиболее ранней доступной точкой кадра.

Поле MAC состоит из идентификатора ключа (Key Identifier), за которым следует дайджест сообщения. Дайджест или криптографическая сумма вычисляется в соответствии с [RFC1321] для заголовка NTP и полей расширения без MAC.

Extension Field n

Формат поля описан в параграфе 7.5. Формат поля расширения NTP.

Key Identifier (keyid)

32-битовое целое число без знака, используемое клиентом и сервером для обозначения секретного 128-битового ключа MD5.

Message Digest (digest)

128-битовое хэш-значение MD5, вычисляемое для ключа, за которым следует заголовок пакета NTP и поля расширения (без учёта Key Identifier и Message Digest).

Следует отметить, что описанный здесь расчёт MAC отличается от заданного в [RFC1305] и [RFC4330], но совместим с генерацией MAC в имеющихся реализациях.

7.4. Пакет Kiss-o'-Death

Если Stratum = 0 (незаданный или недействительный слой), поле Reference Identifier может служить для передачи сообщений, полезных для отчётов о состоянии и контроля доступа. Такие пакеты называются «поцелуем смерти» Kiss-o'-Death или KoD), а сообщения ASCII в них - кодами kiss. Пакеты KoD получили своё имя потому, что сначала их применяли для указания клиенту прекратить передачу пакетов, нарушающих правила доступа к серверу. Коды kiss могут предоставить полезные сведения клиентам NTPv4 и SNTPv4. Kiss-коды представляются 4-символьными строками ASCII (Рисунок 13) с выравниванием по левому краю и дополнением нулями справа. Строки предназначены для символьных дисплеев и log-файлов. Получатель kiss-кода **должен** проверить его и в указанных ниже случаях выполнить действия.

a. Для DENY и RSTR клиент **должен** деактивировать все ассоциации с этим сервером и прекратить отправку пакетов ему.

b. Для kiss-кода RATE клиент **должен** незамедлительно увеличить свой интервал опроса для этого сервера и продолжать увеличение при каждом получении кода RATE[1].

c. Коды, начинающиеся с символа ASCII X, предназначены для экспериментов и разработки и неизвестные значения **должны** игнорироваться.

d. В остальных случаях пакеты KoD не имеют значения для протокола и отбрасываются после проверки.

Код	***Значение***
ACST	Ассоциация относится к unicast-серверу.
AUTH	Отказ при аутентификации сервера.
AUTO	Отказ последовательности Autokey.
BCST	Ассоциация относится к broadcast-серверу.
CRYP	Отказ при криптографической идентификации или аутентификации.

[1] В оригинале ошибочно сказано о снижении интервала. См. https://www.rfc-editor.org/errata/eid3007. *Прим. перев.*

DENY	Отклонение доступа удаленным сервером.
DROP	Потеря партнёра в симметричном режиме.
RSTR	Отклонение доступа в соответствии с локальными правилами.
INIT	Ассоциация ещё не синхронизирована (в первый раз).
MCST	Ассоциация относится к динамически обнаруженному серверу.
NKEY	Ключ не найден (не был установлен или не является доверенным).
RATE	Превышение скорости. Сервер временно отклонил доступ по причине превышения клиентом порога скорости.
RMOT	Изменение ассоциации с удалённого хоста, где работает ntpdc.
STEP	Выполнен шаг изменения системного времени, но ассоциация ещё не ресинхронизирована.

Рисунок 13. Kiss-коды.

Receive Timestamp и Transmit Timestamp (задаёт сервер) не определены для пакетов KoD, на из значения **недопустимо** полагаться и они **должны** отбрасываться.

7.5. Формат поля расширения NTP[1]

В NTPv4 могут включаться поля расширения после заголовка и перед кодом MAC (если MAC присутствует).

Этот документ определяет лишь формат поля, не задавая его использования. Поле расширения содержит сообщение запроса или отклика в формате, показанном на рисунке 14.

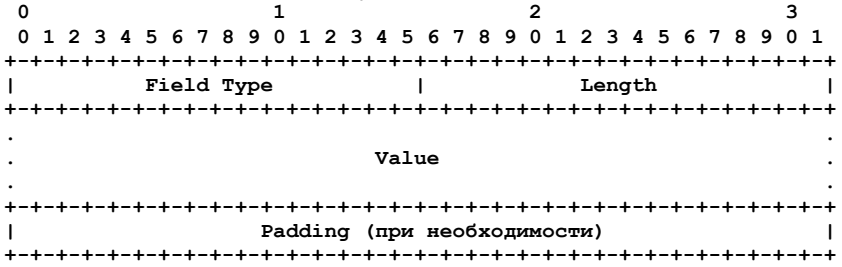

Рисунок 14. Формат поля расширения.

Все поля расширения дополняются нулями до границы слова (4 октета).

Поля Field Type, Value, Padding зависят от заданной функции и не рассматриваются здесь. Значения Field Type заданы в рресстре IANA, а значения Length, Value, Padding определяются указанным в этом реестре документом. Если хост получает поле расширения с неизвестным Field Type, ему **следует** игнорировать это поле расширения и **можно** отбросить пакет совсем, если правила требуют этого.

Хотя минимальный размер поля, содержащего обязательные компоненты, составляет 4 слова (16 октетов), максимальный размер не может превышать 65532 октетов по причине ограниченного размера поля Length.

Length содержит 16-битовое целое число без знака, задающее размер поля расширения в октетах, включая Padding.

7.5.1. Поля расширения и коды MAC

7.5.1.1. Поля расширения при наличии MAC

Поле расширения может использоваться в пакете NTP с кодом MAC, например, как задано в [Autokey]. Спецификация, определяющая новое поле расширения, **должна** указывать, требует ли это поле наличия MAC. Если поле расширения требует MAC, спецификация поля **должна** указывать алгоритм создания и размер MAC. Поле расширения **может** разрешать применение нескольких алгоритмов и в этом случае сведения об использованном алгоритме должны включаться в само поле расширения.

7.5.1.2. Несколько полей расширения с MAC

При наличии нескольких полей расширения, требующих MAC, все эти **должны** требовать один алгоритм и размер MAC. Поля расширения без MAC могут включаться вместе с полями, требующими MAC.

Недопустимо передавать пакет NTP с несколькими полями расширения, требующими различные алгоритмы MAC. Если пакет NTP принят с несколькими полями расширения, распознанными получателем и требующими MAC с разными алгоритмами, пакет **должен** отбрасываться.

7.5.1.3. MAC без полей расширения

Размер MAC более 24 октетов **недопустим** без поля расширения, если длинный код MAC не согласован клиентом и сервером. Согласование может быть выполнено в предыдущем обмене пакетами с полем расширения, задающим размер и алгоритм кодов MAC передаваемых в пакетах NTP.

7.5.1.4. Поля расширения без MAC

При отсутствии MAC одно или несколько полей расширения могут вставляться после заголовка, как указано ниже.

- При одном поле расширения размер этого поля **должен** быть не менее 7 слов (28 октетов).
- При наличии нескольких полей расширения размер последнего поля должен быть не менее 28 октетов, размер других полей расширения **должен** быть не менее 16 октетов для каждого.

8. Протокол в линии

Основой NTP является протокол on-wire, обеспечивающий ядро механизма обмена значениями времени между серверами, партнёрами и клиентами. Протокол по своей природе устойчив к потере и дублированию пакетов. Целостность данных обеспечивается контрольными суммами IP и UDP. Механизмы управления потоком данных и

[1]В оригинале этот параграф содержал ряд ошибок. См. https://www.rfc-editor.org/errata/eid3627. Приведённый здесь текст соответствует корректировке, внесённой RFC 7822. *Прим. перев.*

повторной передачи не требуются и не применяются. Протокол использует метки времени, извлекаемые из заголовков пакетов или системных часов в момент прибытия или отправки пакета. Метки являются прецизионными данными и их следует получать заново в случае повтора передачи на канальном уровне с поправкой на время расчёта MAC.

Сообщения NTP могут использовать два режима связи - «один с одним» или «один со многими», обычно называемые одноадресными (unicast) или широковещательными (broadcast). В этом документе широковещательным считается любой механизм, обеспечивающий передачу от одного ко многим. Для IPv4 это соответствует широковещательной или групповой передаче IPv4, для IPv6 - групповой передаче IPv6. Для этого агентство IANA выделило групповой адрес IPv4 224.0.1.1 и групповой адрес IPv6 с суффиксом :101 и префиксом, определяемым локальными правилами для области действия. Могут применяться также иные групповые адреса в дополнение к указанным.

Протокол в линии (on-wire) использует 4 метки времени с номерами от t1 до t4 и 3 переменных состояния org, rec и xmt, как показано на рисунке 15. Рисунок представляет общий случай, где каждый из двух партнёров (A и B) независимо измеряет смещение и задержку относительно другого. На рисунке метки указаны строчными буквами, а переменные состояния - прописными. Переменные состояния копируются из меток в пакетах при отправке и получении.

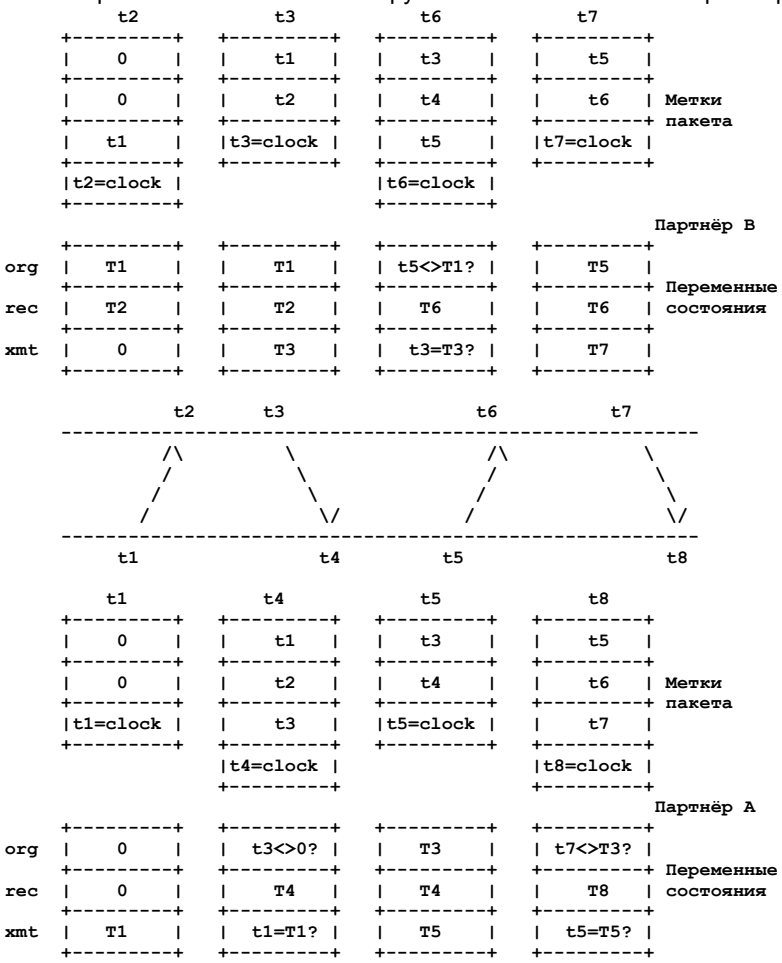

Рисунок 15. Протокол в линии.

На рисунке первый пакет, переданный A, содержит лишь метку источника t1, которая копируется в T1. Партнёр B принимает пакет в момент t2 и копирует t1 в T1 а метку приёма t2 - в T2. В этот момент или несколько позже в t3 партнёр B передаёт A пакет, содержащий t1, t2 и метку отправки t3. Все три метки копируются в соответствующие переменные состояния. A получает в момент t4 пакет с метками t1, t2, t3 и меткой получателя t4. Эти четыре метки служат для расчёта смещения и задержки B относительно A, как описано ниже.

До обновления значений xmt и org выполняются две проверки пригодности для защиты от дубликатов, подделок и повторов. В приведённом выше обмене пакет является дубликатом или повтором, если t3 в нем совпадает с переменной состояния org T3. Пакет является фиктивным, если метка источника t1 в нем не совпадает с переменной состояния xmt T1. В обоих случаях переменные состояния обновляются, а пакет отбрасывается. Для защиты от повторного использования (replay) переданного последним пакета в переменной состояния xmt устанавливает 0 сразу же после обнаружения подделки.

Для 4 наиболее свежих меток времени переменные T1 - T4 служат для расчёта смещения B относительно A

```
theta = T(B) - T(A) = 1/2 * [(T2-T1) + (T4-T3)]
```
[1]

и круговой задержки

```
delta = T(ABA) = (T4-T1) - (T3-T2).
```

Отметим, что выражения в круглых скобках рассчитываются по 64-битовым меткам без знака и будут давать 63 бита значения и бит знака. Это позволяет представлять даты от 68 лет в прошлом до 68 в будущем, но смещение и задержка считаются как сумма и разность величин и дают 62 бита значения и 2 бита знаков, поэтому могут однозначно представлять время от 34 дет в прошлом до 34 в будущем. Иными словами, клиент должен быть установлен в интервале 34 до запуска сервера. Это фундаментальное ограничение 64-битовой арифметики целых чисел.

В реализациях с доступной арифметикой действительных чисел с плавающей запятой разности перевого порядка могут преобразовываться в floating double, а суммы и разности второго порядка рассчитываться с применением этой

[1] В оригинале это уравнение содержало ошибку. См. https://www.rfc-editor.org/errata/eid5020. *Прим. перев.*

арифметики. Поскольку элементы второго порядка обычно очень малы по сравнению со значениями временных меток, здесь нет потери значимости и диапазон однозначных величин восстанавливается до 68 лет (вместо 34).

В некоторых случаях, где начальный сдвиг частоты клиента сравнительно велик, а реальное время распространения мало, результат расчёта задержки может быть отрицательным. Например, если разница частот составляет 100 ppm а интервал T4-T1 - 64 секунды, задержка будет равна -6,4 мсек. Поскольку отрицательные значения в последующих расчётах вводят в заблуждение, значение delta следует фиксировать не меньше s.rho, где s.rho - точность системы в секундах, как описано в параграфе 11.1.

Приведённое выше обсуждение предполагает наиболее общий случай, где два симметричных партнёра независимо измеряют смещения и задержки между собой. В случае сервера, не хранящего состояний протокол можно упростить. Такой сервер копирует T3 и T4 из клиентского пакета в T1 и T2 серверного пакета и присоединяет метку отправки T3 до передачи пакета клиенту. Сведения о заполнении других полей протокола приведены в разделе 9 и Приложении.

Отметим, что описанный протокол on-wire устойчив к повторному использованию пакетов отклика от сервера. Однако он не стоек к повторному использованию клиентских запросов, которые будут вынуждать сервер повторить пакет с новыми значениями T2 и T3, дающими некорректные результаты для смещения и задержки. Этой уязвимости можно избежать, установив значение 0 для переменной состояния xmt после расчёта смещения и задержки.

9. Партнёрский процесс

Последующие описания процессов включают важные переменные состояния и обзор операций, реализованных в подпрограммах. В тексте часто указываются ссылки на скелет программы в приложении, включающий фрагменты кода C, более подробно описывающие функции. Код включает параметры, переменные и объявления, требуемые для соответствующей спецификации реализации NTPv4. Однако в работающей реализации может потребоваться ещё много переменных.

Партнерский процесс вызывается прибытием пакета от сервера или партнёра. Процесс запускает протокол on-wire для определения смещения часов и круговой задержки, а также рассчитывает статистику, применяемую системным процессом и процессом опроса. Переменные партнёра создаются в структуре данных ассоциации, когда та инициализируется, и обновляются при получении пакетов. Для каждого сервера имеется партнёрский процесс, процесс опроса и процесс ассоциации.

9.1. Переменные партнёрского процесса

На рисунках 16 - 19 приведены базовые имена, имена формул и краткие описания переменных партнёра. Базовые имена и имена формул взаимозаменяемы, имена формул предназначены для удобочитаемости уравнений в спецификации. Если явно не указано иное, для партнёрских переменных применяется префикс p.

Имя	***Формула***	***Описание***
srcaddr	srcaddr	Адрес источника
srcport	srcport	Порт источника
dstaddr	dstaddr	Адрес получателя
dstport	destport	Порт получателя
keyid	keyid	Идентификатор ключа

Рисунок 16. Переменные конфигурации партнёрского процесса.

Имя	***Формула***	***Описание***
leap	leap	Индикатор високосных секунд
version	version	Номер версии
mode	mode	Режим
stratum	stratum	Слой
ppoll	ppoll	Показатель опроса
rootdelay	delta_r	Корневая задержка
rootdisp	epsilon_r	Корневая дисперсия
refid	refid	Идентификатор источника (эталона)
reftime	reftime	Аременная метка эталона

Рисунок 17. Переменные партнёрского процесса.

Имя	***Формула***	***Описание***
org	T1	Временная метка источника
rec	T2	Временная метка приёма
xmt	T3	Временная метка передачи
t	t	Время пакета

Рисунок 18. Переменные метки времени партнёрского процесса.

Имя	***Формула***	***Описание***
offset	theta	Смещение часов
delay	delta	Круговая задержка
disp	epsilon	Дисперсия
jitter	psi	Вариации (jitter)
filter	filter	Фильтр часов
tp	t_p	Время фильтра

Рисунок 19. Статистические переменные партнёрского процесса.

Ниже указаны переменные конфигурации, обычно инициализируемые при активизации ассоциации из конфигурационного файла или по прибытии первого пакета для неизвестно ассоциации.

srcaddr

IP-адрес удалённого сервера или эталонных часов. Он будет адресом получателя для пакетов из ассоциации.

srcport
Номер порта UDP на сервере или эталонных часах. Он будет портом получателя для пакетов из ассоциации. При работе в симметричных режимах (1 и 2), это поле должно содержать номер порт NTP PORT (123), выделенный IANA. В иных режимах номер порта определяется локальными правилами.

dstaddr
IP-адрес клиента. Он будет адресом отправителя для пакетов из ассоциации.

dstport
Номер порта UDP на клиенте, исходно порт NTP PORT (123), выделенный IANA. Он будет портом отправителя для пакетов из ассоциации.

keyid
Идентификатор симметричного ключа для 128-битового ключа MD5, служащий для генерации и проверки MAC. Клиент и сервер или партнёр могут использовать разные значения, но они должны указывать один ключ.

Переменные с рисунка 17 обновляются из заголовка каждого прибывающего пакета и интерпретируются так же, как одноименные переменные пакета. Для последующей обработки удобно преобразовать короткий формат NTP для r.rootdelay и r.rootdisp в переменные партнёра с форматом floating double.

Переменные с рисунка 18 включают временные метки, передаваемые протоколом (раздел 8). Переменная t указывает счётчик секунд c.t, связанный с этими значениями. Переменная c.t поддерживается процессом корректировки часов, описанным в разделе 12. Она учитывает секунды с момента запуска службы. Переменные с рисунка 19 включают статистику, рассчитываемую подпрограммой clock_filter(), описанной в разделе 10. Переменная tp содержит число секунд, связанное с этими значениями.

9.2. Операции партнёрского процесса

Процедура приёма определяет поток процессов по прибытию пакета, как показано функцией receive() в Приложении A.5.1. Для управления доступом не нужен специальный метода, хотя реализациям рекомендуется включать схему, аналогичную многим широко применяемым. Функция access() в Приложении A.5.4 описывает метод реализации ограничений на основе списка контроля доступа (access control list или ACL). Проверка формата требует корректного размера и выравнивания полей, приемлемого номера версии (1-4) и корректного синтаксиса полей расширения при их наличии.

Конкретные требования по проверке подлинности не задаются, однако при поддержке аутентификации требуется реализация алгоритма хэширования MD5, описанного в [RFC1321].

Затем выполняется поиск в таблице ассоциаций по адреса и порту источника, например, с помощью функции find_assoc() из Приложения A.5.1. На рисунке 20 приведена таблица диспетчеризации, где столбцы соответствуют режиму пакетов, а строки - режиму ассоциации. Ячейка на пересечении указывает один из описанных ниже шагов.

		Пакетный режим				
Режим ассоциации		***1***	***2***	***3***	***4***	***5***
Нет ассоциации	0	NEWPS	DSCRD	FXMIT	MANY	NEWBC
Симметричная активная	1	PROC	PROC	DSCRD	DSCRD	DSCRD
Симметричная пассивная	2	PROC	ERR	DSCRD	DSCRD	DSCRD
Клиент	3	DSCRD	DSCRD	DSCRD	PROC	DSCRD
Сервер	4	DSCRD	DSCRD	DSCRD	DSCRD	DSCRD
Широковещательная	5	DSCRD	DSCRD	DSCRD	DSCRD	DSCRD
Широковещательный клиент	6	DSCRD	DSCRD	DSCRD	DSCRD	PROC

Рисунок 20. Таблица диспетчеризации партнёров.

DSCRD
Некритическое нарушение протокола в результате программной ошибки, долгой задержки или повтора пакета. Партнерский процесс отбрасывает пакет и завершается.

ERR
Критическое нарушение протокола в результате программной ошибки, долгой задержки или повтора пакета. Партнерский процесс отбрасывает пакет, деактивирует симметричныую пассивную ассоциацию и завершается.

FXMIT
Клиентский пакет (режим 3) не соответствует ассоциации (режим 0). Если адрес получателя не является широковещательным, сервер создаёт серверный пакет (режим 4) и возвращает его клиенту без сохранения состояния. Создаётся заголовок серверного пакета, пример описан в функции fast_xmit() Приложения A.5.3. Заголовок собирается из полученного пакета и системных переменных, как показано на рисунке 21. Если системные переменные хранятся в форме floating double, они должны преобразовываться в короткие метки NTP.

```
Переменная пакета -->     Переменная
r.leap            -->     p.leap
r.mode            -->     p.mode
r.stratum         -->     p.stratum
r.poll            -->     p.ppoll
r.rootdelay       -->     p.rootdelay
r.rootdisp        -->     p.rootdisp
r.refid           -->     p.refid
r.reftime         -->     p.reftime
r.keyid           -->     p.keyid
```

Рисунок 21. Приём заголовка пакета.

Отметим, что при отказе аутентификации сервер возвращает специальное сообщение crypto-NAK, включающее обычные данные заголовка NTP (Рисунок 8), но с кодом MAC, содержащим 4 октета нулей. Клиент **может** принять или отвергнуть данные сообщения. После этого партнёрский процесс завершается.

Если адрес получателя является групповым, отправитель работает в клиентском режиме manycast. Если пакет действителен и слой сервера меньше слоя клиента, сервер передаёт обычный среверный пакет (режим 4), но с одним из индивидуальных (unicast) адресов получателя. При отказе аутентификации crypto-NAK не передаётся. После этого партнёрский процесс завершается.

MANY

Серверный пакет (режим 4) не соответствует ассоциации. Обычно это происходит при отклике manycast-сервера на переданный ранее групповой пакет клиента. Если пакет действителен, активируется обычная ассоциация клиента (режим 3) и работа продолжается как при активации ассоциации по конфигурационному файлу.

NEWBC

Широковещательный пакет (режим 5) не соответствует ассоциации. Клиент активирует обычную (режим 3) или широковещательную (режим 6) ассоциацию. Примеры показаны в функциях mobilize() и clear() Приложения A.2. Затем пакет проверяется и инициализируются переменные партнёра, как показано в функции packet() Приложения A.5.1.1.

Если реализация не поддерживает дополнительных функций защиты или калибровки, ассоциация устанавливается в режим широковещательного клиента (режим 6) и процесс партнёра завершается. Реализации с аутентификацией на основе открытых ключей **может** применять Autokey или эквивалентный протокол защиты. Реализациям **следует** устанавливать режим ассоциации 3 и запускать короткий обмен клиент-сервер для определения задержки расппространения. После обмена устанавливается режим ассоциации 6 и процесс партнёра продолжается в режиме «только прослушивание». Отметим, различие между пакетом режима 6, зарезервированным для функций мониторинга и управления NTP, и режимом 6 в ассоциации.

NEWPS

Пакет симметричного активного режима (режим 1) не соответствует ассоциации. Клиент активирует симметричныую пассивную ассоциацию (режим 2). Пример приведён в функциях mobilize() и clear() Приложения A.2. Обработка продолжается, как описано ниже для PROC.

PROC

Пакет соответствует имеющейся ассоциации. Метки времени в пакете аккупатно проверяются для обнаружения недействительных, дублированных и фиктивных пакетов. Дополнительные проверки показаны на рисунке 22. Отметим, что все пакеты, включая crypto-NAK, считаются действительными лишь после прохождения этих тестов.

Тип пакета	***Описание***
1 Дубликат пакета	В лучшем случае пакет является дубликатом, в худшем - повтором (replay) от хакера. Это возможно в симметричном режиме при неравномерных интервалах опроса.
2 Фиктивный пакет	
3 Недействительный пакет	Одно или несколько полей меток времени недействительно. Это обычно происходит в симметричном режиме, когда один партнёр передал пакет другому, а тот ещё не получил свой первый отклик.
4 Доступ отвергнут	Адрес источникам запрещён контролем доступа.
5 Отказ аутентификации	Криптографический дайджест сообщения на соответствует MAC.
6 Нет синхронизации	Сервер не синхронизирован с действительным источником.
7 Неверные данные в заголовке	Одно или несколько полей заголовка недействительно.

Рисунок 22. Проверка ошибок пакета.

Обработка продолжается копированием переменных пакета в переменные партнёра, как показано на рисунке 21. Пример этого приведён в функции packet() Приложения A.5.1.1. Функция receive() реализует тесты 1-5 (Рисунок 22), packet() - тесты 6-7. При обнаружении ошибки пакет отбрасывается и процесс партнёра завершается.

Протокол on-wire вычисляет смещение часов theta и круговую задержку delta по 4 наиболее свежим меткам времени, как описано в разделе 8. Хотя, в принципе, можно выполнить все расчёты, кроме разности первого порядка для временных меток, в арифметике с фиксированной запятой, много проще преобразовать разности первого порядка в floating double и выполнить оставшиеся расчёты в этой арифметике, как предполагается ниже.

Затем используется 8-битовый регистр сдвига p.reach (раздел 13) для определения доступности сервера и актуальности данных. Регистр сдвигается на 1 бит влево при передаче пакета с установкой правого бита в 0. При поступлении действительного пакета правый бит устанавливается (1). При наличии в регистре отличных от 0 битов сервер считается доступным, в ином случае - недоступным. Поскольку интервал опроса у партнёра мог измениться с последнего пакета, интервал опроса у хоста пересматривается. Пример представлен в функции poll_update() Приложения A.5.7.2.

Статистика дисперсии epsilon(t) представляет максимальную ошибку в результате допуска частоты и времени с момента передачи последнего пакета. Она инициализируется значением epsilon(t_0) = r.rho + s.rho + PHI * (T4-T1), когда измерение выполняется в момент t_0 согласно счетчику секунд. Здесь r.rho - точность пакета (параграф 7.3), а s.rho - точность системы (параграф 11.1), выраженные в секундах. Эти элементы требуются для учёта погрешности при считывании системных часов сервера и клиента.

Затем дисперсия растёт с постоянной скоростью PHI, т. е. в момент t будет epsilon(t) = epsilon(t_0) + PHI * (t-t_0). С принятым по умолчанию допуском PHI = 15 ppm это составляет около 1,3 сек/сутки. При таком подходе аргумент t отбрасывается и дисперсия представляется просто как epsilon. Остальная статистика рассчитывается алгоритмом фильтрации часов, описанным в следующем параграфе.

10. Алгоритм фильтрации часов

Алгоритм фильтра часов является частью процесса партнёра и обрабатывает поток данных для выбора образцов, представляющих наиболее точное время. Алгоритм создаёт переменные, показанные на рисунке 19, включая смещение (theta), задержку (delta), дисперсию (epsilon), вариации (psi) и время прибытия (t). Эти данные применяются алгоритмами смягчения для определения лучшего и окончательного смещения, применяемого дисциплиной системных часов, а также для определения работоспособности сервера и его пригодности для синхронизации.

Алгоритм фильтра часов сохраняет наиболее свежие квартеты выборки (theta, delta, epsilon, t) в структуре фильтра, служащей 8-ступенчатым регистром сдвига. Квартеты сохраняются в порядке прибытия пакетов, t указывает время прибытия по счётчику секунд и его не следует путать с партнёрской переменной tp.

Для обеспечения фильтру достаточного числа выборок и удаления старых данных применяется описанная далее схема. Исходно для квартетов всех ступеней устанавливаются фиктивные значения (0, MAXDISP, MAXDISP, 0), по мере прибытия новых пакетов квартеты сдвигаются в регистре с отбрасыванием старых и в результате остаются лишь действительные квартеты. Если 3 младших бита регистра охвата имеют значение 0, показывающее, что истекли 3 интервала опроса без приёма новых действительных пакетов, процесс опроса вызывает алгоритм фильтра часов с фиктивным квартетом, как будто тот прибыл из сети. Если это сохраняется в течение 8 интервалов опроса, регистр возвращается в исходное состояние. На следующем этапе ступени регистра сдвига копируются во временный список с

сортировкой по возрастанию delta. Путь i - индекс ступени, начинающейся с минимального значения delta. Если первая эпоха квартета t_0 не позднее эпохи последней действительной выборки tp, программа завершается без изменения текущих переменных партнёра. В ином случае для дисперсии i-й записи epsilon_i - выражение

```
                 i=n-1
                 ---     epsilon_i
epsilon =         \     ----------
                  /        (i+1)
                 ---     2
                 i=0
```

задаёт дисперсию партнёра p.disp. Отметим, что при переполнении epsilon на входе или выходе фильтра часов смысл должен быть ясен из контекста.

Наблюдатель следует отметить: (a) если все все ступени содержат фиктивные квартеты с дисперсией MAXDISP, рассчитанная дисперсия составит чуть меньше 16 сек, (b) при каждом сдвиге действительного квартета в регистр дисперсия падает чуть меньше, чем на половину в зависимости от дисперсии действительного квартета, (c) после четвёртого действительного квартета дисперсия обычно будет чуть меньше 1 сек, что является предполагаемым значением параметра MAXDIST по которому алгоритм выбора определяет приемлемость переменных партнёра.

Пусть смещение первой ступени в сортированном списке будет theta_0. Тогда для других ступеней в любом порядке вариация будет определяться средним значением RMS[1]

```
          +-----                                -----+^1/2
          |              n-1                        |
          |              ---                        |
          |    1         \                    2     |
psi   = | -------- *     /    (theta_0-theta_j)     |
          |  (n-1)       ---                        |
          |              j=1                        |
          +-----                                -----+
```

где n - число действительных квартетов в фильтре (n > 1). Для обеспечения согласованности и предотвращения исключений при делении в других расчётах значение psi ограничивается снизу точностью системы s.rho, выраженной в секундах. Хотя обычно вариации не считаются основным фактором ранжирования качества серверов, это важный индикатор работы хронометража и состояний перегрузки в сети. Особое значение для алгоритмов смягчения имеет дистанция синхронизации с партнёром, которая вычисляется по задержке и дисперсии.

```
lambda = (delta / 2) + epsilon
```

Отметим, что epsilon и, следовательно, lambda растёт со скоростью PHI. Переменная lambda не является переменной состояния, поскольку она рассчитывается при каждом применении. Это часть корневой дистанции синхронизации, применяемая алгоритмами смягчения как показатель качества времени, доступного от каждого сервера.

Важно отметить, что, в отличие от NTPv3, ассоциации NTPv4 не указывают тайм-аут установкой слоя 16 и индикатора високосных секунд 3. Переменные ассоциации сохраняют значения, полученные при поступлении последнего пакета. В NTPv4 значение lambda растёт со временем, поэтому в конечном итоге дистанция синхронизации превышает порог MAXDIST, после чего ассоциация считается не пригодной для синхронизации.

Пример реализации алгоритма фильтра часов содержится в функции clock_filter() из Приложения A.5.2.

11. Системный процесс

При получении каждой новой выборки (theta, delta, epsilon, jitter, t) алгоритмом фильтра часов все процессы партнёров сканируются алгоритмами смягчения, состоящими из алгоритмов выбора, кластера, комбинирования и дисциплины часов в системном процессе. Алгоритм выбора сканирует все ассоциации и отбрасывает фальшивые часы, которые указывают неверное время, оставляя лишь истинные. В серии раундов алгоритм кластера отбрасывает ассоциации, наиболее удалённые статистически от центроида, пока не останется заданное минимальное их число. Алгоритм комбинирования даёт лучшую и окончательную статистику на основе средневзвешенного значения. Окончательное смещение передаётся алгоритму дисциплины часов для корректировки времени системных часов.

Алгоритм кластера выбирает одного из оставшихся в качестве партнёра системы. Связанная с ним статистика (theta, delta, epsilon, jitter, t) служит для создания системных переменных, наследуемых зависимыми серверами и клиентами, доступными для других приложений на той же машине.

11.1. Переменные системного процесса

На рисунке 23 приведены базовые имена, формулы и краткие описания каждой системной переменной. Если явно не указано иное, все переменные имеют суффикс s.

Имя	*Формула*	*Описание*
t	t	Время обновления
p	p	Идентификатор партнёра системы
leap	leap	Индикатор високосной секунды
stratum	stratum	Слой (stratum)
precision	rho	Точность
offset	THETA	Комбинированное смещение
jitter	PSI	Комбинированные вариации
rootdelay	DELTA	Корневая задержка
rootdisp	EPSILON	Корневая дисперсия
v	v	Список оставшихся
refid	refid	Идентификатор эталона
reftime	reftime	Эталонное время
NMIN	3	Минимальное число остающихся
CMIN	1	Минимальное число кандидатов

Рисунок 23. Переменные системного процесса.

[1] В оригинале приведённое ниже выражение содержало ошибку. См. https://www.rfc-editor.org/errata/eid5600. *Прим. перев.*

За исключением переменных t, p, offset, jitter и констант NMIN и CMIN, для переменных системы применяется тот же формат и интерпретация, что и для одноимённых переменных партнёра. Параметры NMIN и CMIN, используемые алгоритмами выбора и кластера, описаны в следующем параграфе.

Переменная t указывает счётчик секунд в момент последнего обновления, пример приведён в функции clock_update() (Приложение A.5.5.4). Переменная p содержит идентификатор партнёра системы, определяемый функцией cluster() (параграф 11.2.2). Переменная precisionc имеет такой же формат, что и одноимённая переменная партнёра. Точность определяется как двоичных логарифм большего из значений разрешения и времени считывания. Например, точность тактового генератора от сети переменного тока 60 Гц составляет 16 мсек, даже если аппаратные часы представляются с точностью в наносекунду. Переменные offset и jitter определяет алгоритм комбинирования (параграф 11.2.3). Они представляют лучшее и окончательное смещение и вариацию, используемые дисциплиной системных часов.

Исходно все переменные сбрасываются в 0, затем устанавливается leap = 3 (не синхронизировано) и stratum = MAXSTRAT (16). Помните, что MAXSTRAT в передаваемых пакетах указывается значением 0.

11.2. Операции системного процесса

На рисунке 24 показаны операции системного процесса, выполняемые программой выбора часов. Описанный в параграфе 11.2.1 алгоритм выдаёт набор основных предполагаемых кандидатов (истинные часы) на основе принципов соглашения. Алгоритм кластера (параграф 11.2.2) отбрасывает лишних кандидатов для создания более узкого набора. Алгоритм комбинирования (11.2.3) выдаёт наилучшее и окончательное смещение для алгоритма дисциплины часов. Пример представлен в Приложении A.5.5.6. Если алгоритм выбора не может создать набор кандидатов (majority clique) или не возможно выдать хотя бы CMIN оставшихся, системный процесс завершается без дисциплины системных часов. При успешном выполнении алгоритм кластера выбирает статистически оучшего кандидата как партнёра системы и его переменные наследуются как переменные системы.

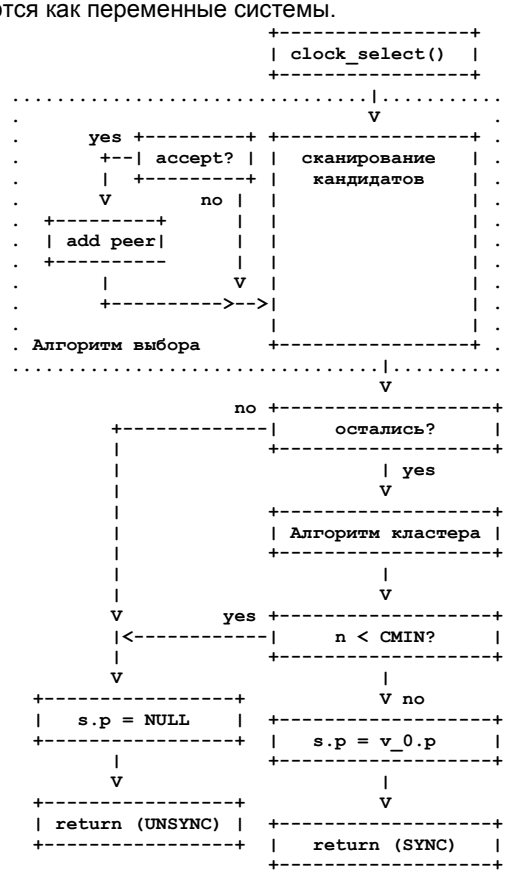

Рисунок 24. Процедура выбора часов.

11.2.1. Алгоритм выбора

Алгоритмы выбора и кластера описаны по отдельности, но в скелете кода объединены. Алгоритм выбора ищет пересечение интервалов, содержащее основные истинные часы, по принципам византийского алгоритма, исходно предложенного Marzullo [ref6], но изменённого для повышения точности. Обзор алгоритма приведён ниже, а код представлен в первой половине функции clock_select() в Приложении A.5.5.1.

Сначала серверы, не подходящие по правилам протокола обнаруживаются и отбрасываются, как показано в функции accept() Приложения A.5.2[1]. Затем создаётся набор триплетов (p, type, edge) для оставшихся кандидатов. Здесь p - идентификатор ассоциации, type указывает верхнюю (+1), среднюю (0) и нижнюю (-1) конечные точки интервала корректности кандидата с центром theta. Это даёт 3 триплета lowpoint (p, -1, theta - lambda), midpoint (p, 0, theta), highpoint (p, +1, theta + lambda), где lambda - корневая дистанция синхронизации. Пример расчёта показан в функции root_dist() Приложения A.5.1.2[2]. Этапы алгоритма приведены ниже.

1. Для каждой из m ассоциаций 3 триплета помещаются в список кандидатов, как указано выше.
2. Триплеты списка сортируются по краевым компонентам в порядке lowpoint, midpoint, highpoint. Для числа фальшивых часов устанавливается значение f = 0.
3. Устанавливается число средних точек d = 0, а также c = 0 и выполняется сканирование от низшей точки к высшей. Значение c увеличивается на 1 для каждой lowpoint, снижается на 1 для каждой highpoint и

[1]В оригинале ошибочно указано A.5.5.3. См. https://www.rfc-editor.org/errata/eid3132. *Прим. перев.*
[2]В оригинале это предложение содержало ошибки. См. https://www.rfc-editor.org/errata/eid4019. *Прим. перев.*

добавляется 1 к каждой midpoint. Если c >= m - f, выполнение завершается и в качестве l устанавливается текущая нижняя точка.

4. Устанавливается c = 0 и выполняется сканирование от верхней точки к нижней. К c добавляется 1 для каждой highpoint и вычитается 1 для каждой lowpoint, к d добавляется 1 для каждой . Если c >= m - f, выполнение завершается и в качестве u устанавливается текущая верхняя точка.

5. Проверяется выполнение условий d <= f[1] и l < u. При соблюдении выполняется 5A, иначе - 5B.

   5A. Интервалом пересечения будет [l, u].

   5B. К f добавляется 1 и проверяется условие f < (m / 2). При выполнении возврат к п. 3, иначе - 6.

6. Отказ - набор кандидатов не найден, поскольку они не подходят для дисциплины системных часов.

Алгоритм подробно описан в Приложении A.5.5.1. Отметим, что он начинается с допущения отсутствия фальшивых часов (f = 0) и попытки еайти непустой интервал пересечения со средними точками всех корректных серверов (истинные часы). Если такой интервал не найден, число предполагаемых ложных часов увеличивается на 1 и попытка повторяется. Если интервал найден и число ложных часов меньше числа истинных, набор кандидатов найден и midpoint каждых истинных часов (theta) представляет кандидатов, доступных для алгоритма кластера.

Если набор кандидатов не найден или число истинных часов меньше CMIN, кандидатов для дисциплины системных часов не достаточно. CMIN задаёт минимальное число серверов, соответствующих требованиям корректности. Сомнительные операторы будут устанавливать CMIN для обеспечения множества избыточных серверов, доступных для надлежащего смягчения. Однако по историческим причинам для CMIN по умолчанию установлено значение 1.

11.2.2. Алгоритм кластера

Кандидаты из набора помещаются в список оставшихся v в форме триплетов (p, theta_p, psi_p, lambda_p), где p - идентификатор ассоциации, theta_p, psi_p и stratum_p - текущее смещение, вариации (jitter) и слой ассоциации p, соответственно, а lambda_p - показатель качества, равный stratum_p * MAXDIST + lambda, где lambda - корневая дистанция синхронизации для p. Список обрабатывается алгоритмом кластера, как описано ниже, а пример можно найти во второй половине алгоритма clock_select() из Приложения A.5.5.1.

1. Пусть (p, theta_p, psi_p, lambda_p) представляет сохранившегося кандидата.

2. Кандидаты сортируются по росту lambda_p. Пусть n - число кандидатов, а NMIN - минимально требуемое число оставшихся.

3. Для каждого кандидата рассчитывается вариация выбора как

```
        +-----                          -----+^1/2
        |        n-1                         |
        |        ---                         |
        |   1    \                       2   |
psi_s = | ---- * /   (theta_s - theta_j)     |
        |  n-1   ---                         |
        |        j=1                         |
        +-----                          -----+
```

4. Выбирается psi_max как кандидат с наибольшим значением psi_s.

5. Выбирается psi_min как кандидат с наименьшим значением psi_p.

6. Если psi_max < psi_min или n <= NMIN, выполняется 6A, иначе - 6B.

   6A. Алгоритм завершён, оставшиеся в списке кандидаты ранжируются в порядке предпочтения. Первый элемент списка представляет партнёра системы, переменные которого позже служат для обновления системных переменных.

   6B. Удаляется кандидат с psi_max, значение n уменьшается на 1 и алгоритм возвращается к п. 3.

Алгоритм работает с серией раундов, в каждом из которых отбрасывается статистический выброс с максимальной вариацией выбора psi_s. Однако, если psi_s меньше минимальной вариации партнёра psi_p, отбрасывание выбросов уже не даст улучшения. Это минимальное число оставшихся даёт условие завершения алгоритма. По завершении финальное значение psi_max сохраняется как вариация выбора системы PSI_s для последующего применения.

11.2.3. Алгоритм комбинирования

Маршрут комбинирования часов обрабатывает оставшихся кандидатов для создания лучших и окончательных данных для алгоритма дисциплины часов. Программа обрабатывает статистику смещения и вариаций партнёра для создания комбинированного смещения THETA и вариаций PSI_p, где статистика каждого сервера «взвешивается» обратной величиной корневой дистанции синхронизации и результат нормализуется (см. функцию clock_combine() в Приложении A.5.5.5)

Комбинированное значение THETA передаётся программе обновления часов. Первый кандидат из списка оставшихся назначается партнёром системы с идентификатором p. Его значение является компонентом вариаций системы PSI и применяется вместе с PSI_s для расчёта, как показано ниже.

```
PSI = [(PSI_s)^2 + (PSI_p)^2]^1/2
```

При каждом обновлении, полученном от партнёра системы вызывается программа обновления часов. По правилу обновление отбрасывается, если время прибытия p.t не строго позже последнего принятого обновления s.t. Метки IGNOR, PANIC, ADJ, STEP указывают коды возврата локальной программы часов, описанной в следующем параграфе.

IGNORE указывает, что обновление проигнорировано, как выброс, PANIC означает смещение, превышающее порог PANICT (1000 сек) и по нему **следует** завершать программу с диагностическим сообщением для журнала системы (log). STEP указывает, что смещение меньше PANICT, но больше порога ступени STEPT (125 мсек). В этом случае

[1] В оригинале ошибочно указано d = f. См. https://www.rfc-editor.org/errata/eid4366. *Прим. перев.*

часы переводятся на корректное смещение, но, поскольку это означает недействительность всех данных партнёра, все ассоциации **должны** быть сброшены, а клиент начинает заново как при исходном запуске.

ADJ означает смещение меньше порога ступени и действительное обновление. В этом случае системные переменные обновляются по переменным партнёра, как показано на рисунке 25.

```
Системная переменная <-- Системная переменная партнёра
         s.leap      <-- p.leap
         s.stratum   <-- p.stratum + 1
         s.offset    <-- THETA
         s.jitter    <-- PSI
         s.rootdelay <-- p.delta_r + delta
         s.rootdisp  <-- p.epsilon_r + p.epsilon + 5* p.psi + PHI * (t_s - p.t) + |THETA|
         s.refid     <-- p.refid
         s.reftime   <-- p.reftime
         s.t         <-- p.t
```

Рисунок 25. Обновление системных переменных.

Переменная t_s указывает время обновления системных переменных. На рисунке не показана важная деталь. Инкремент дисперсии (p.epsilon + 5 * p.psi + PHI * (t_s - p.t) + |THETA|) ограничен снизу значением MINDISP. В подсетях с очень быстрыми процессорами и скоростью, а также очень малой задержкой и дисперсией это ведёт к монотонно-определённому росту s.rootdisp (EPSILON), что предотвращает петли между партнёрами, работающими в одном слое[1].

Системные переменные доступны зависимым прикладным программам как номинальная статистика производительности. Системное смещение THETA - смещение часов относительно источников синхронизации, системные вариации PSI - оценка ошибки при определении значения, иногда называемая ожидаемой ошибкой. Корневая задержка DELTA - суммарная круговая задержка до первичного сервера. Корневая дисперсия EPSILON - дисперсия аккумулированная в сети от первичного сервера. Дистанция синхронизации определяется выражением

```
LAMBDA = EPSILON + DELTA / 2
```

которое указывает максимальную ошибку, обусловленную всеми причинами, и называемую дистанцией синхронизации от корня.

Пример программы обновления часов представлен в Приложении A.5.5.4.

11.3. Алгоритм дисциплины часов

Алгоритм дисциплины часов NTPv4, сокращенно называемый дисциплиной, работает как комбинация двух достаточно различающихся систем обратной связи. В схеме фазовой автоподстройки частоты (phase-locked loop или PLL) периодически обновляется фаза с интервалами в несколько микросекунд для минимизации временной ошибки напрямую и ошибки частоты - опосредованно. В контуре автоподстройки частоты (frequency-locked loop или FLL) периодически обновляется частота с микросекундными интервалами для минимизации ошибки частоты напрямую и ошибки времени - опосредованно. Как показано в [ref7], PLL работает лучше при доминировании сетевых вариаций, а FLL - при доминировании дрейфа тактового генератора. В этом разделе описана работа NTPv4, подробное описание принципов устройства дано в [ref7] вместе с анализом производительности.

Дисциплина реализована как система управления с обратной связью (Рисунок 26). Переменная theta_r представляет смещение алгоритма комбинирования (опорная фаза), а theta_c - смещение VFO (фаза управления). Каждое обновление создаёт сигнал V_d, представляющий мгновенную разность фаз theta_r - theta_c. Фильтр часов для каждого сервера работает как линия задержки с ответвлениями, при этом вывод берётся с ответвления, выбранного алгоритмом фильтра часов. Алгоритмы выбора, кластера и комбинирования объединяют данные нескольких фильтров для создания сигнала V_s. Кольцевой фильтр с импульсным откликом F(t) создаёт сигнал V_c, управляющий частотой VFO omega_c и, таким образом, фазой theta_c, замыкая контур. Сигнал V_c генерируется процессом корректировки часов (раздел 12). Уравнения, реализующие эти функции, представлены в Приложениях A.5.5.6 и A.5.6.1.

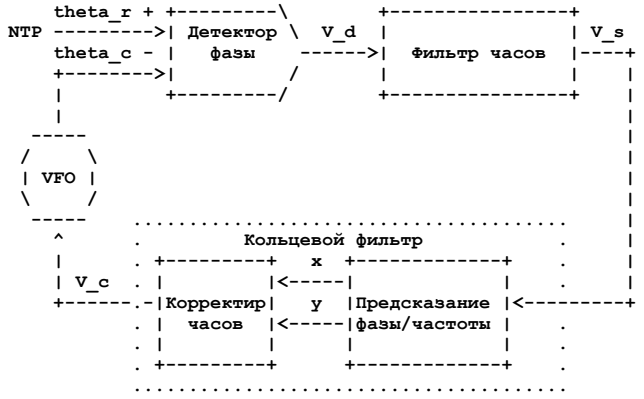

Рисунок 26. Контуры обратной связи дисциплины часов.

Исходно описанный выше псевдолинейный контур обратной связи работает для дисциплины системных часов. Однако в некоторых случаях нелинейный алгоритм обеспечивает существенное улучшение. Одним из таких случаев является запуск дисциплины без знания частоты внутренних часов. Псевдолинейному контуру нужно несколько часов для точного измерения и в большую часть этого времени интервал измерений нельзя увеличить. Описанный ниже нелинейный контур делает это за 15 минут. Другим случаем являются случайные пики вариаций при наличии перегрузки в каналах сети. Описанный ниже конечный автомат устойчив к пикам ошибок длительностью менее 55 минут.

На рисунке 27 приведены переменные и параметры, имена формул и краткие описания. Если явно не указано иное, для всех переменных предполагается префикс c. Переменные t, tc, state, hyster, count являются целочисленными, остальные - floating double. Назначение каждой переменной приведено ниже в описании алгоритма.

Имя Формула Описание

[1]В оригинале этот абзац и рисунок 25 содержали ошибки. См. https://www.rfc-editor.org/errata/eid5601 и https://www.rfc-editor.org/errata/eid5604. *Прим. перв.*

t	timer	Счётчик секунд
offset	theta	Комбинированное смещение
resid	theta_r	Оставшееся смещение
freq	phi	Частота часов
jitter	psi	Вариации смещения часов
wander	omega	Дрейф частоты часов
tc	tau	Постоянная времени (log2)
state	state	Состояние
adj	adj	Корректировка частоты
hyster	hyster	Счётчик гистерезиса
STEPT	125	Порог ступени (0,125 сек)
WATCH	900	Порог шага (сек)
PANICT	1000	Порог «паники» (1000 сек)
LIMIT	30	Предел гистерезиса
PGATE	4	Порог гистерезиса (hysteresis gate)
TC	16	Масштаб постоянной времени
AVG	8	Постоянная усреднения

Рисунок 27. Переменные и параметры дисциплины часов.

Процесс прерывается незамедлительно, если смещение превышает порог PANICT (1000 сек). Программа смены состояния rstclock() представлена в Приложении A.5.5.7, а на рисунке 28 показаны используемые в ней функции смены состояний. В столбцах таблицы приведены имена состояний, условие и действие для случаев, кода смещение theta меньше порога ступени и больше этого порога, а также комментарии.

Статус	*theta < STEP*	*theta > STEP*	*Комментарий*
NSET	->FREQ adjust time	->FREQ step time	Нет файла частоты
FSET	->SYNC adjust time	->SYNC step time	Файл частоты
SPIK	->SYNC adjust freq adjust time	if < 900 s ->SPIK else ->SYNC step freq step time	Обнаружен выброс (пик)
FREQ	if < 900 s ->FREQ else ->SYNC step freq adjust time	if < 900 s ->FREQ else ->SYNC step freq adjust time	Исходная частота
SYNC	->SYNC adjust freq adjust time	if < 900 s ->SPIK else ->SYNC step freq step time	Нормальная работа

Рисунок 28. Функции смены состояния.

В таблице следующее состояние указывается стрелкой ->, а далее приведены действия. Такие операции как корректировка времени и частоты реализуются контуром обратной связи PLL/FLL в функции local_clock(). Операция «сдвига часов на шаг» (step clock) реализуется установкой часов напрямую, но это происходит лишь после порога порога WATCH (900 сек), когда смещение превышает порог шага STEPT (0,125 сек). Это позволяет избежать ненужных корректировок в случае перегрузки в сети.

Статистика вариаций (psi) и дрейфа (omega) рассчитывается с использованием экспоненциального среднего значения с весовым коэффициентом AVG. Показатель постоянной времени (tau) определяется сравнением psi с величиной текущего смещения theta. Если смещение больше PGATE (4), умноженного на вариации часов, счётчик гистерезиса hyster уменьшается на 2, а ином случае - увеличивается на 1. Если hyster увеличивается до верхнего предела LIMIT (30), tau увеличивается на 1, при снижении до нижнего предела -LIMIT (-30) значение tau уменьшается на 1. Обычно tau колеблется около MAXPOLL, но быстро снижается, если скачок температуры вызывает всплеск частоты.

12. Процесс корректировки часов

Фактический процесс корректировки часов выполняется с интервалом в 1 секунду для учёта корректировки частоты и фиксированного процента оставшегося смещения theta_r. По сути, theta_r представляет экспоненциальное затухание значения theta, создаваемого фильтром контура обратной связи при каждом обновлении. Параметр TC для удобства масштабирует постоянную времени в соответствии с интервалом опроса. Отметим, что дисперсия EPSILON растёт на величину PHI каждую секунду.

Процесс корректировки часов включает прерывание по таймеру, управляющее счётчиком секунд c.t. Отсчёт начинается с 0 при запуске службы и значение инкрементируется 1 раз в секунду. При каждом прерывании вызывается функция clock_adjust() для включения времени дисциплины часов и корректировки частоты, затем ассоциации сканируются, чтобы сравнить значение счётчика секунд с переменной состояния p.next, определённой в следующем параграфе. Если значение счётчика не меньше p.next, вызывается процесс опроса и рассчитывается следующее значение p.next.

Пример процесса корректировки часов содержится в функции clock_adjust() Приложения A.5.6.1.

13. Процесс опроса

Каждая ассоциация поддерживает процесс опроса, запускаемый регулярно для сосздания и отправки пакетов в симметричных, клиентских и широковещательных серверных ассоциациях. Процесс работает непрерывно, независимо от доступности серверов, для поддержки фильтра часов и регистра доступа.

13.1. Переменные процесса опроса

На рисунке 29 приведены имена, формулы и краткие описания переменных и параметров процесса опроса. Если явно не указано иное, для переменных предполагается суффикс p.

Имя	*Формула*	*Описание*
hpoll	hpoll	Показатель опроса

last	last	Время последнего опроса
next	next	Время следующего опроса
reach	reach	Регистр доступа
unreach	unreach	Счётчик недоступности
UNREACH	24	Предел недоступности
BCOUNT	8	Счётчик пиков
BURST	flag	Флаг режима burst
IBURST	flag	Флаг разрешения iburst

Рисунок 29. Переменные и параметры процесса опроса.

Переменные процесса опроса выделяются в структуре данных вместе с переменными процесса партнёра. Ниже приведены описания переменных, а параметры рассматриваются позднее.

hpoll
Целое число со знаком, представляющую показатель опроса (двоичный логарифм числа секунд).
last
Целочисленное значение счётчика секунд с момент передачи последнего пакета.
next
Целое число секунд до момента передачи следующего пакета.
reach
8-битовый целочисленный регистр сдвига, совместно используемые процессами партнёра и опроса.
unreach
Целое число секунд, в течение которых сервер был недоступен.

13.2. Операции процесса опроса

Как описано выше, процесс корректировки часов вызывается 1 раз в секунду и сам вызывает процедуру опроса для каждой ассоциации. Если время отправки следующего опросного сообщения больше значения счётчика секунд, программа сразу же завершается. Симметричные (режимы 1, 2), клиентские (режим 3) и широковещательные серверный (режим 5) ассоциации регулярно отправляют пакеты. Широковещательные ассоциации клиентов (режим 6) запускают процедуру для обновления переменных reach и unreach, но не передают пакеты. Процесс опроса вызывает процесс передачи для отправки пакета. В случае burst > 0 ничего не делается, кроме вызова процедуры обновления опроса для установки интервала следующего опроса. В ином случае переменная reach сдвигается влево на 1 бит с установкой для правого бита значения 0. Если сервер не был слышен в течение трёх последних интервалов опроса, вызывается программа фильтра часов для увеличения дисперсии (Приложение A.5.7.3).

Если флаг BURST установлен, сервер доступен и имеется действительный источник синхронизации, клиент передаёт блок (burst) из BCOUNT (8) пакетов за каждый интервал опроса. Интервал между пакетами в блоке составляет 2 секунды. Это полезно для точного измерения вариаций с длинными интервалами опроса. Если флаг IBURST установлен и это первый пакет, переданный с момента недоступности сервера, клиент передаёт блок пакетов. Это полезно для быстрого сокращения дистанции синхронизации до значения ниже порога и синхронизации часов.

Если флаг P_MANY установлен в слове ассоциации p.flags, это ассоциация является клиентской многоадресной (manycast). Такая ассоциация передаёт клиентские пакеты по назначенному групповому адресу с интервалом MINPOLL, начиная работу с TTL = 1. Если к моменту следующего опроса активировано (mobilized) менее MINCLOCK серверов, значение TTL увеличивается на 1. Если значение достигло TTLMAX и не найдено MINCLOCK серверов, интервал опроса увеличивается до достижения BEACON, после чего процесс возобновляется с начала.

Функция poll() включает увеличение интервала опроса, если сервер становится недоступным. Если reach имеет ненулевое значение, сервер доступен и устанавливается unreach = 0, в ином случае значение unreach увеличивается на 1 для каждого опроса вплоть до UNREACH. После этого hpoll для каждого опроса увеличивается на 1, что удваивает интервал опроса, вплоть до значения MAXPOLL, определяемого функцией poll_update(). Когда сервер снова становится доступным, устанавливается unreach = 0, для hpoll устанавливается значение системной переменной tc и работа возобновляется в обычном режиме.

Пакет передаётся процессом отправки. Некоторые поля заголовка копируются из системных переменных, оставленных предыдущим пакетом, а другие - из системных переменных. На рисунке 30 показано, какие значения копируются в каждое поле заголовка. В реализациях, применяющих данные типа floating double для корневой задержки и дисперсии, эти значения должны преобразовываться в короткий формат NTP. Остальные поля копируются без изменения из переменных партнёра и системы или извлекаются как метки из часов системы.

```
Переменная пакета <--    Переменная
x.leap            <--    s.leap
x.version         <--    s.version
x.mode            <--    s.mode
x.stratum         <--    s.stratum
x.poll            <--    s.poll
x.precision       <--    s.precision
x.rootdelay       <--    s.rootdelay
x.rootdisp        <--    s.rootdisp
x.refid           <--    s.refid
x.reftime         <--    s.reftime
x.org             <--    p.xmt
x.rec             <--    p.dst
x.xmt             <--    clock
x.keyid           <--    p.keyid
x.digest          <--    md5 digest
```

Рисунок 30. Заголовок xmit_packet.

Процедура обновления опроса вызывается при получении действительного пакета и сразу же после отправки опросного сообщения. В блочном (burst) режиме фиксируется интервал опроса 2 сек, в ином случае для показателя hpoll устанавливается меньшее из значений ppoll для последнего полученного пакета и hpoll из функции опроса, но не меньше MINPOLL и не больше MAXPOLL. Таким образом дисциплина часов может применяться излишне часто (oversampled) но не слишком редко. Это нужно для сохранения динамики подсети и защиты от ошибок протокола.

Показатель опроса преобразуется в целое число, которое при добавлении к последней переменной времени опроса, задаёт значение времени следующего опроса. В конечном итоге для переменной времени последнего опроса устанавливается текущее значение счётчика секунд.

14. Простой протокол сетевого времени (SNTP)

Первичным серверам и клиентам, соответствующим подмножеству NTP, называемому простым протоколом сетевого времени (Simple Network Time Protocol или SNTPv4) [RFC4330], не требуется реализовать алгоритмы смягчения, описанные в разделе 9 и последующих разделах этого документа. Протокол SNTP предназначен для первичных серверов, оборудованных одними эталонными часами, и клиентов с одним восходящим сервером без зависимых клиентов. Полная реализация NTPv4 предназначена для вторичных серверов с несколькими восходящими серверами и множеством нисходящих серверов или клиентов. За исключением этого, серверы и клиенты NTP и SNTP полностью совместимы и могут перемешиваться в подсетях NTP.

Первичный сервер SNTP, реализующий протокол on-wire, описанный в разделе 8 , не имеет восходящих серверов за исключением одних эталонных часов. В принципе, он неотличим от первичного сервера NTP, имеющего алгоритмы смягчения и за счёт этого способного смягчать различия между несколькими эталонными часами.

При получении запроса от клиента первичный сервер SNTP создаёт и отправляет пакет отклика, показанный на рисунке 31. Отметим, что поле дисперсии в заголовке должно обновляться в соответствии с разделом 5.

```
Переменная пакета <--     Переменная
x.leap            <--     s.leap
x.version         <--     r.version
x.mode            <--     4
x.stratum         <--     s.stratum
x.poll            <--     r.poll
x.precision       <--     s.precision
x.rootdelay       <--     s.rootdelay
x.rootdisp        <--     s.rootdisp
x.refid           <--     s.refid
x.reftime         <--     s.reftime
x.org             <--     r.xmt
x.rec             <--     r.dst
x.xmt             <--     clock
x.keyid           <--     r.keyid
x.digest          <--     md5 digest
```

Рисунок 31. Заголовок fast_xmit.

Клиент SNTP, реализующий протокол on-wire, имеет 1 сервер и не имеет зависимых клиентов. Он может работать с любым подмножеством протокола NTP on-wire, а самый простой подход использует лишь метку времени передачи от сервера и игнорирует прочие поля. Однако добавочные сложности полной реализации протокола on-wire минимальны и такая реализация приветствуется.

15. Вопросы безопасности

Требования безопасности для NTP более строги, чем для большинства распределенных служб. Во-первых, операции механизмов проверки подлинности и синхронизации времени неразрывно связаны между собой. Для надёжной синхронизации времени требуются криптографические ключи, которые действительны лишь в течение заданного срока, но временные интервалы можно применять лишь при надёжной синхронизации участвующих клиентов и серверов с UTC. Кроме того, подсеть NTP по своей природе имеет иерархию, поэтому время и доверие передаются от первичных серверов в корне иерархии к клиентам (листьям) через вторичные серверы.

Клиент NTP может заявлять наличие истинного времени зависимым приложениям лишь в том случае, когда все серверы на пути к первичным аутентифицированы. В NTP каждый сервер аутентифицирует серверы слоя (stratum) с меньшим номером и (по индукции) серверы с наименьшим значением stratum (первичные). Важно отметить, что проверка подлинности в контексте NTP не обязательно предполагает корректность времени. Клиент NTP активирует множество одновременных ассоциаций с разными серверами и использует специально созданный алгоритм согласования для выбора истинных часов (truechimer) из популяции, возможно включающей ложные часы (falseticker).

Спецификация NTP предполагает, что цель злоумышленника состоит во внедрении ложных значений времени, нарушении работы протокола и засорение сети, серверов или клиентов фиктивными пакетами, которые истощают ресурсы и препятствуют обслуживанию легитимных приложений. В архитектуру, протокол и алгоритмы NTP уже включено множество механизмов защиты. Схема обмена метками времени в линии по своей природе устойчива к подменам, потерям и повторному использованию пакетов в атаках. Фильтр часов, алгоритмы выбора и кластеризации разработаны для защиты от ложных кандидатов (evil clique) в византийских атаках. Хотя алгоритмы и сопутствующие проверки работоспособности не предназначены специально для борьбы со злоумышленниками, они хороши работают в течение многих лет, отклоняя некорректно работающие, но предположительно дружественные варианты. Однако эти механизмы не обеспечивают безопасной идентификации и аутентификации серверов для клиента. Без дополнительной защиты злоумышленник может организовать любую из указанных ниже атак.

1. Злоумышленник может перехватывать и архивировать пакеты и открытые значения, создаваемые и передаваемые через сеть.
2. Атакующий может создавать пакеты быстрее, чем сервер, сеть или клиент способны обрабатывать их, особенно при использовании ресурсоёмких криптографических расчётов.
3. В атаках с прослушиванием злоумышленник может перехватывать, изменять и повторно использовать пакеты. Однако он не может навсегда предотвратить передачу исходного пакета, т. е. прервать линию, а способен лишь исказить или перегрузить обмен пакетами. В общем случае изменённые пакеты не могут приходить к жертве раньше исходных, а также у злоумышленника не может быть секретных ключей и параметров отождествления сервера.
4. При перехвате с участием человека или маскировании пакетов злоумышленник размещается между сервером и клиентом, поэтому он может перехватывать, изменять и повторно использовать пакеты, а также препятствовать передаче исходных пакетов. Однако у него не может быть секретных ключей и параметров отождествления сервера.

Модель безопасности NTP предполагает ряд ограничений, указанных ниже.

1. Время работы алгоритмов с открытым ключом сравнительно велико и сильно варьируется. Как правило, производительность функций синхронизации часов существенно снижается при использовании такого алгоритма для каждого пакета NTP.
2. В некоторых режимах работы серверу нереально сохранять переменные состояния для каждого клиента. Однако возможно создать их заново по прибытии пакета от этого клиента.
3. Срок действия криптографических значений должен соблюдаться, для чего требуется надёжность системных часов. Однако источники синхронизации для системных часов должны быть доверенными. Такая циклическая зависимость функций хронометража требует особого внимания.
4. Функции защиты клиенты должны включать лишь открытые значения, передаваемые через сеть. Секретный значения не должны покидать машину, где они созданы, за исключением случаев использования специальных доверенных агентов (trusted agent или TA), предназначенных для таких целей.

В отличие от модели безопасности Secure Shell (SSH), где клиент должен пройти безопасную аутентификацию на сервере, в NTP сервер должен пройти проверку подлинности у клиента. В SSH каждый адрес интерфейса можно связать со своим именем, возвращаемым запросом reverse-DNS. В этой схеме могут требоваться разные пары ключей «открытый-секретный» для каждого интерфейса со своим именем. Очевидным преимуществом такого подхода является возможность организации своего «отсека» безопасности для каждого интерфейса. Это позволяет межсетевому экрану, например, требовать аутентификацию клиента на одних интерфейсах, не требуя на других.

В случае NTP, как указано здесь, широковещательные клиенты NTP уязвимы из-за некорректного поведения или враждебности широковещательных серверов SNTP и NTP в сети Internet. Такая опасность может быть сведена к минимуму несколькими способами. Можно организовать фильтрацию, чтобы клиенты NTP получали доступ лишь к доверенным серверам NTP, что предотвратит попадание вредоносного трафика к клиентам NTP. Криптографическая аутентификация у клиента позволит применять при синхронизации часов лишь должным образом подписанные сообщения NTP. Более надёжную аутентификацию может обеспечить механизм Autokey [RFC5906].

В разделе 8 рассмотрены возможные угрозы, связанные с повторным использованием клиентских запросов. Соблюдение рекомендаций этого раздела поможет защититься от таких атак.

Следует отметить, что эта спецификация описывает имеющуюся реализацию. Хотя недостатки защиты алгоритма MD5 хорошо известны, его применение в спецификации NTP согласуется с широким применением в сообществе Internet.

16. Взаимодействие с IANA

Порт UDP/TCP 123 был ранее выделен IANA для этого протокола. Агентство IANA выделило групповой адрес IPv4 224.0.1.1 и групповой адрес IPv6, заканчивающийся на :101 для протокола NTP. Этот документ добавляет поля расширения NTP, позволяющие разработку будущих расширений протокола, где конкретное расширение задаётся субполем Field Type в поле расширения. Агенство IANA создало и поддерживает реестр для типов полей расширения (Extension Field Types), связанных с этим протоколом, который исходно пуст. По мере необходимости у будущем могут определяться новые типы полей расширения, регистрируемые по процедуре IETF Review [RFC5226].

Агентство IANA создало новый реестр для кодов идентификации эталогов NTP (NTP Reference Identifier codes), включающий значения, заданные в параграфе 7.3. Новые коды могут добавляться по процедуре First-Come-First-Serve (FCFS). Формат реестра показан ниже.

ID	*Источник времени*
GOES	Geosynchronous Orbit Environment Satellite
GPS	Global Position System
...	...

Рисунок 32. Коды идентификации эталонных источников.

Агентство IANA создало новый реестр для кодов NTP Kiss-o'-Death. Реестр включает коды, заданные в параграфе 7.4, и может расширяться по процедуре FCFS. Формат реестра показан ниже.

Код	*Значение*
ACST	Ассоциация относится к unicast-серверу.
AUTH	Отказ при аутентификации сервера.
...	...

Рисунок 33. Коды Kiss.

Для идентификаторов эталонов и кодов Kiss-o'-Death, начинающихся с символа X (эксперименты и разработка) запрашивать IANA не требуется.

17. Благодарности

Авторы благодарны Karen O'Donoghue, Brian Haberman, Greg Dowd, Mark Elliot, Harlan Stenn, Yaakov Stein, Stewart Bryant, Danny Mayer за технические рецензии и текстовый вклад в этот документ.

18. Литература

18.1. Нормативные документы

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

18.2. Дополнительная литература

[CGPM] Bureau International des Poids et Mesures, "Comptes Rendus de la 15e CGPM", 1976.

[ITU-R_TF.460] International Telecommunications Union, "ITU-R TF.460 Standard-frequency and time-signal emissions", February 2002.

[RFC1305] Mills, D., "Network Time Protocol (Version 3) Specification, Implementation and Analysis", RFC 1305, March 1992.

[RFC1345] Simonsen, K., "Character Mnemonics and Character Sets", RFC 1345, June 1992.

[RFC4330] Mills, D., "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI", RFC 4330, January 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5906] Haberman, B., Ed. and D. Mills, "Network Time Protocol Version 4: Autokey Specification", RFC 5906, June 2010.

[ref6] Marzullo and S. Owicki, "Maintaining the time in a distributed system", ACM Operating Systems Review 19, July 1985.

[ref7] Mills, D.L., "Computer Network Time Synchronization - the Network Time Protocol", CRC Press, 304 pp, 2006.

[ref9] Mills, D.L., Electrical and Computer Engineering Technical Report 06-6-1, NDSS, June 2006, "Network Time Protocol Version 4 Reference and Implementation Guide", 2006.

Приложение A. Скелет кода

Это приложение предназначено для описания протокола и алгоритмов реализации в общем виде с использованием так называемой скелетной программы. Она состоит из набора определений, структур и фрагментов кода, показывающих основные операции протокола без сложностей, присущих фактической реализации протокола. Программа не предназначена для реального применения.

Здесь включено большинство возможностей эталонной реализации, но нет предоставления эталонных часов и криптографии с открытым ключом (Autokey). Отсутствует фильтр huff-n'-puff filter, гистерезис anti-clockhop и процедуры мониторинга. Многие из значений, которые могут меняться в эталонной реализации, здесь предполагаются константами. Пакеты kiss-o'-death представлены минимально и без соответствующего кода.

Программа не обеспечивает компактности и скорости, она предназначена лишь для демонстрации алгоритмов с достаточной для понимания их работы точностью. Скелет кода состоит из 8 сегментов - сегмент заголовка, включаемый во все остальные сегменты, сегмент кода основной программы, интерфейсы с ядром, системой ввода-вывода и системными часами, а также процессы партнёра, системы, опроса и clock_adjust. Они представлены вместе с определениями и переменными, характерными для каждого процесса.

A.1. Глобальные определения

A.1.1. Определения, константы, параметры

```
#include <math.h>               /* избегать жалоб на sqrt() */
#include <sys/time.h>           /* для gettimeofday() и т. п. */
#include <stdlib.h>             /* для malloc() и т. п. */
#include <string.h>             /* для memset() */

/*
 * Типы данных
 *
 * Эта программа предполагает тип int размеров 32 бита и long - 64 бита.
 * В большинстве расчётов применяется естественный тип floating double.
 * Типы данных в некоторых полях заголовков требуют преобразования.
 * Некоторые поля заголовков делятся по октетам и представлены здесь
 * октетами.
 *
 * 64-битовый формат меток времени NTP, применяемый в расчётах меток,
 * является беззнаковым числом секунд слева от бита 32. Для этих значений
 * разрешена лишь операция вычитания, дающая значения со знаком и размером
 * 31 бит. Короткий формат 32-битовых меток NTP применяется при расчёте
 * задержки и дисперсии. Значение целых секунд указывается слева от бита
 * 16. Для этих значений разрешено лишь сложение и умножение на константу.
 *
 * Адреса IPv4 задаются 32 битами, IPv6 - 128 битами. Поле дайджеста
 * сообщения содержит 128 битов результата расчёта MD5. Поля точности и
 * интервала опроса задаются двоичным логарифмом (log2) числа секунд.
 */
typedef unsigned long long tstamp;   /* Формат метки времени NTP */
typedef unsigned int tdist;     /* Короткий формат NTP */
typedef unsigned long ipaddr;   /* Адрес IPv4 или IPv6 */
typedef unsigned long digest;   /* Дайджест md5 */
typedef signed char s_char;     /* Точность или интервал опроса (log2) */

/*
 * Макросы преобразования меток времени
 */
```

```
#define FRIC        65536.                  /* 2^16 как double */
#define D2FP(r)     ((tdist)((r) * FRIC))   /* короткая метка NTP */
#define FP2D(r)     ((double)(r) / FRIC)

#define FRAC       4294967296.              /* 2^32 как double */
#define D2LFP(a)    ((tstamp)((a) * FRAC))  /* метка NTP */
#define LFP2D(a)    ((double)(a) / FRAC)
#define U2LFP(a)    (((unsigned long long) \
                      ((a).tv_sec + JAN_1970) << 32) + \
                      (unsigned long long) \
                      ((a).tv_usec / 1e6 * FRAC))

/*
 * Арифметические преобразования
 */
#define LOG2D(a)        ((a) < 0 ? 1. / (1L << -(a)) : \
                            1L << (a))          /* опрос и т. п. */
#define SQUARE(x)       ((x) * (x))[1]
#define SQRT(x)         (sqrt(x))

/*
 * Глобальные константы. Часть их может конвертироваться в переменные,
 * которые можно менять в конфигурации или рассчитывать «на лету».
 * Например, эталонная реализация рассчитывает PRECISION на лету и
 * обеспечивает настройку для определений, помеченных %.
 */
#define VERSION         4       /* номер версии */
#define MINDISP         .01     /* % минимальная дисперсия (s) */
#define MAXDISP         16      /* максимальная дисперсия (s) */
#define MAXDIST         1       /* % порог дистанции (s) */
#define NOSYNC          0x3     /* високосные секунды не синхронизированы */
#define MAXSTRAT        16      /* Максимальный номер слоя (бесконечность) */
#define MINPOLL         6       /* % минимальный интервал опроса (64 сек)*/
#define MAXPOLL         17      /* % максимальный интервал опроса (36,4 час) */
#define MINCLOCK        3       /* минимальное число оставшихся manycast */
#define MAXCLOCK        10      /* максимальное число кандидатов manycast */
#define TTLMAX          8       /* max ttl manycast */
#define BEACON          15      /* максимальный интервал между beacon */

#define PHI             15e-6   /* % допуск частоты (15 ppm) */
#define NSTAGE          8       /* ступени регистра часов */
#define NMAX            50      /* максимальное число партнёров */
#define NSANE           1       /* % минимальное пересечение оставшихся */
#define NMIN            3       /* % минимум оставшихся для кластера */

/*
 * Глобальные значения возврата
 */
#define TRUE            1       /* boolean true */
#define FALSE           0       /* boolean false */

/*
 * Коды возврата локального процесса часов
 */
#define IGNORE          0       /* игнорировать */
#define SLEW            1       /* настройка поправки */
#define STEP            2       /* настройка шага */
#define PANIC           3       /* паника - нет корректировки */

/*
 * Флаги системы
 */
#define S_FLAGS         0       /* любые системные флаги */
#define S_BCSTENAB      0x1     /* включение широковещательного клиента */

/*
 * Флаги партнёра
 */
#define P_FLAGS         0       /* любые флаги партнёра */
#define P_EPHEM         0x01    /* ассоциация временна */
#define P_BURST         0x02    /* блоки (burst) разрешены */
#define P_IBURST        0x04    /* начальный блок разрешён */
#define P_NOTRUST       0x08    /* аутентифицированный доступ */
#define P_NOPEER        0x10    /* аутентифицированная активизация */
#define P_MANY          0x20    /* manycast-клиент */

/*
 * Коды проверки подлинности
 */
#define A_NONE          0       /* без аутентификации */
#define A_OK            1       /* успешная аутентификация */
#define A_ERROR         2       /* ошибка аутентификации */
#define A_CRYPTO        3       /* crypto-NAK */
```

[1] В оригинале ошибочно указано (x * x). См. https://www.rfc-editor.org/errata/eid3404. *Прим. перев.*

```
/*
 * Коды состояния ассоциаций
 */
#define X_INIT          0       /* инициализация */
#define X_STALE         1       /* тайм-аут */
#define X_STEP          2       /* шаг времени */
#define X_ERROR         3       /* ошибка аутентификации */
#define X_CRYPTO        4       /* получено сообщение crypto-NAK */
#define X_NKEY          5       /* недоверенный ключ */

/*
 * Определения режимов протокола
 */
#define M_RSVD          0       /* резерв */
#define M_SACT          1       /* симметричный активный */
#define M_PASV          2       /* симметричный пассивный */
#define M_CLNT          3       /* клиент */
#define M_SERV          4       /* сервер */
#define M_BCST          5       /* широковещательный сервер */
#define M_BCLN          6       /* широковещательный клиент */

/*
 * Определения состояний часов
 */
#define NSET            0       /* часы не устанавливались */
#define FSET            1       /* частота установлена из файла */
#define SPIK            2       /* обнаружен всплеск (spike) */
#define FREQ            3       /* режим частоты */
#define SYNC            4       /* часы синхронизированы */

#define min(a, b)   ((a) < (b) ? (a) : (b))
#define max(a, b)   ((a) < (b) ? (b) : (a))
```

A.1.2. Структуры данных пакета

```
/*
 * Принимаемые и передаваемые пакеты могут включать необязательный код
 * MAC, состоящий из идентификатора ключа keyid и дайджеста сообщения
 * (mac в принимаемой структуре и dgst - в передаваемой). NTPv4
 * поддерживает необязательные поля расширения между заголовком и MAC,
 * но здесь это не описано.
 *
 * Приёмный пакет.
 *
 * Отметим, что метка времени dst не является частью самого пакета. Она
 * извлекается по прибытии и возвращается в приёмный буфер вместе с его
 * размером и данными. Некоторые из полей char сжимаются в заголовке, но
 * здесь это не рассматривается.
 */
struct r {
        ipaddr  srcaddr;        /* Адрес источника (удалённый) */
        ipaddr  dstaddr;        /* Адрес получателя (локальный) */
        char    version;        /* Номер версии */
        char    leap;           /* Индикатор високосных секунд */
        char    mode;           /* Режим */
        char    stratum;        /* Слой (stratum) */
        char    poll;           /* Интервал опроса */
        s_char  precision;      /* Точность */
        tdist   rootdelay;      /* Корневая задержка */
        tdist   rootdisp;       /* Корневая дисперсия */
        char    refid;          /* Идентификатор эталона */
        tstamp  reftime;        /* Время эталона */
        tstamp  org;            /* Метка времени источника */
        tstamp  rec;            /* Метка времени приёма */
        tstamp  xmt;            /* Метка времени передачи */
        int     keyid;          /* Идентификатор ключа */
        digest  mac;            /* Дайджест сообщения */
        tstamp  dst;            /* Метка времени получателя */
} r;

/*
 * Передаваемый пакет
 */
struct x {
        ipaddr  dstaddr;        /* Адрес источника (локальный) */
        ipaddr  srcaddr;        /* Адрес получателя (удалённый) */
        char    version;        /* Номер версии */
        char    leap;           /* Индикатор високосных секунд */
        char    mode;           /* Режим */
        char    stratum;        /* Слой (stratum) */
        char    poll;           /* Интервал опроса */
        s_char  precision;      /* Точность */
        tdist   rootdelay;      /* Корневая задержка */
        tdist   rootdisp;       /* Корневая дисперсия */
        char    refid;          /* Идентификатор эталона */
        tstamp  reftime;        /* Время эталона */
        tstamp  org;            /* Метка времени источника */
```

```
        tstamp  rec;            /* Метка времени приёма */
        tstamp  xmt;            /* Метка времени передачи */
        int     keyid;          /* Идентификатор ключа */
        digest  dgst;           /* Дайджест сообщения */
} x;
```

A.1.3. Структуры данных ассоциации

```
   /*
    * Структура ступени фильтра. Отметим, что t в этой и других структурах
    * указывает время процесса, а не реальное время. Время процесса
    * увеличивается на 1 каждую секунду реального времени.
    */
   struct f {
           tstamp  t;               /* Время обновления */
           double  offset;          /* Смещение часов */
           double  delay;           /* Круговая задержка */
           double  disp;            /* Дисперсия */
   } f;

   /*
    * Структура ассоциации используемая процессами партнёра и опроса.
    */
   struct p {

           /*
            * Переменные, задаваемые в конфигурации.
            */
           ipaddr  srcaddr;         /* Адрес источника (удалённый) */
           ipaddr  dstaddr;         /* Адрес получателя (локальный) */
           char    version;         /* Номер версии */
           char    hmode;           /* Режим хоста */
           int     keyid;           /* Идентификатор ключа */
           int     flags;           /* Флаги опций */

           /*
            * Переменные, устанавливаемые принятым пакетом.
            */
           char    leap;            /* Индикатор високосных секунд */
           char    pmode;           /* Режим партнёра */
           char    stratum;         /* Слой (stratum) */
           char    ppoll;           /* Интервал опроса партнёра */
           double  rootdelay;       /* Корневая задержка */
           double  rootdisp;        /* Корневая дисперсия */
           char    refid;           /* Идентификатор эталона */
           tstamp  reftime;         /* Время эталона */
   #define begin_clear org          /* Начало чистой области */
           tstamp  org;             /* Метка времени источника */
           tstamp  rec;             /* Метка времени приёма */
           tstamp  xmt;             /* Метка времени передачи */

           /*
            * Расчётные данные
            */
           double  t;               /* Время обновления */
           struct f f[NSTAGE];      /* Фильтр часов */
           double  offset;          /* Смещение партнёра */
           double  delay;           /* Задержка партнёра */
           double  disp;            /* Дисперсия партнёра */
           double  jitter;          /* RMS для вариаций */

           /*
            * Переменные процесса опроса
            */
           char    hpoll;           /* Интервал опроса для хоста */
           int     burst;           /* Счётчик блоков (burst) */
           int     reach;           /* Регистр доступа */
           int     ttl;             /* ttl (manycast) */
   #define end_clear unreach        /* Конец чистой области */
           int     unreach;         /* Счётчик недоступности */
           int     outdate;         /* Время последнего опроса */
           int     nextdate;        /* Время следующего опроса */
   } p;
```

A.1.4. Структуры данных системы

```
   /*
    * Список, применяемый алгоритмом пересечения.
    */
   struct m {                       /* m применяется для Marzullo */
           struct p *p;             /* Указатель на структуру партнёра */
           int     type;            /* high +1, mid 0, low -1 */
           double  edge;            /* Граница интервала корректности */
   } m;

   /*
    * Список оставшихся, применяемый алгоритмом кластера.
```

```
 */
struct v {
        struct p *p;            /* Указатель на структуру партнёра */
        double  metric;         /* Показатель сортировки */
} v;

/*
 * Системная структура
 */
struct s {
        tstamp  t;              /* Время обновления */
        char    leap;           /* Индикатор високосных секунд */
        char    stratum;        /* Слой (stratum) */
        char    poll;           /* Интервал опроса */
        char    precision;      /* Точность */
        double  rootdelay;      /* Корневая задержка */
        double  rootdisp;       /* Корневая дисперсия */
        char    refid;          /* Идентификатор эталона */
        tstamp  reftime;        /* Время эталона */
        struct m m[NMAX];       /* Список кандидатов */
        struct v v[NMAX];       /* Список оставшихся */
        struct p *p;            /* Идентификатор ассоциации */
        double  offset;         /* Комбинированное смещение */
        double  jitter;         /* Комбинированные вариации */
        int     flags;          /* Флаги опций */
        int     n;              /* Число оставшихся */
} s;
```

A.1.5. Структуры данных локальных часов

```
/*
 * Структура локальных часов
 */
struct c {
        tstamp  t;              /* Время обновления */
        int     state;          /* Текущее состояние*/
        double  offset;         /* Текущее смещение */
        double  last;           /* Предыдущее смещение*/
        int     count;          /* Счётчик качаний (jiggle) */
        double  freq;           /* Частота */
        double  jitter;         /* RMS для вариаций */
        double  wander;         /* RMS для дрейфа */
} c;
```

A.1.6. Прототипы функций

```
/*
 * Процесс партнёра
 */
void    receive(struct r *);    /* Приём пакета */
void    packet(struct p *, struct r *); /* Обработка пакета */
void    clock_filter(struct p *, double, double, double); /* Фильтр */
double  root_dist(struct p *);  /* Расчёт корневой дистанции */
int     fit(struct p *);        /* Определение пригодности сервера */
void    clear(struct p *, int); /* Очистка ассоциации */
int     access(struct r *);     /* Определение ограничений для доступа */

/*
 * Системный процесс
 */
int     main();                 /* Основная программа */
void    clock_select();         /* Поиск лучших часов */
void    clock_update(struct p *); /* Обновление системных часов */
void    clock_combine();        /* Комбинирование смещений */

/*
 * Процесс локальных часов
 */
int     local_clock(struct p *, double); /* Дисциплина часов */
void    rstclock(int, double, double); /* Смена состояния часов */

/*
 * Процесс корректировки часов
 */
void    clock_adjust();         /* Процесс 1-секундного таймера */

/*
 * Процесс опроса
 */
void    poll(struct p *);               /* Процесс опроса */
void    poll_update(struct p *, int); /* Обновление интервала опроса */
void    peer_xmit(struct p *);  /* Передача пакета */
void    fast_xmit(struct r *, int, int); /* Передача пакета отклика */

/*
 * Вспомогательные функции
 */
```

```
  digest  md5(int);               /* Создание дайджеста сообщения */
  struct p *mobilize(ipaddr, ipaddr, int, int, int, int); /* Активация */
  struct p *find_assoc(struct r *); /* Поиск в таблице ассоциаций */

  /*
   * Интерфейс с ядром
   */
  struct r *recv_packet();        /* Ожидание пакета */
  void    xmit_packet(struct x *); /* Передача пакета */
  void    step_time(double);      /* Шаг времени */
  void    adjust_time(double);    /* Корректировка (slew) времени */
  tstamp  get_time();             /* Считывание времени */
```

A.2. Основная программа и утилиты

```
/*
 * определения
 */
#define PRECISION       -18     /* Точность (log2 s)  */
#define IPADDR          0       /* Любой адрес IP */
#define MODE            0       /* Любой режим NTP */
#define KEYID           0       /* Любой идентификатор ключа */

/*
 * main() - основная программа
 */
int
main()
{
        struct p *p;            /* Указатель на структуру партнёра */
        struct r *r;            /* Указатель на принятый пакет */

        /*
         * Чтение опций команды и инициализация переменных системы.
         * Эталонная реализация оценивает точность каждой машины, измеряя
         * приращение времени при считывании системных часов.
         */
        memset(&s, sizeof(s), 0);
        s.leap = NOSYNC;
        s.stratum = MAXSTRAT;
        s.poll = MINPOLL;
        s.precision = PRECISION;
        s.p = NULL;

        /*
         * Инициализация переменных локальных часов.
         */
        memset(&c, 0, sizeof(c));[1]
        if (/* файл частоты */ 0) {
                c.freq = /* freq */ 0;
                rstclock(FSET, 0, 0);
        } else {
                rstclock(NSET, 0, 0);
        }
        c.jitter = LOG2D(s.precision);

        /*
         * Чтение файла конфигурации и активация постоянных ассоциаций
         * с заданными адресами, версией, режимом, ключами и флагами.
         */
        while (/* активация настроенной ассоциации */ 0) {
                p = mobilize(IPADDR, IPADDR, VERSION, MODE, KEYID,
                    P_FLAGS);
        }

        /*
         * Запуск системного таймера, увеличивающегося 1 раз в секунду,
         * чтение прибывающих пакетов, создание меток приёма и вызов
         * функции receive().
         */
        while (0) {
                r = recv_packet();
                r->dst = get_time();
                receive(r);
        }

        return(0);
}

/*
 * mobilize() - активация и инициализация ассоциации.
 */
struct p
*mobilize(
        ipaddr  srcaddr,        /* IP-адрес отправителя */
```

[1] В оригинале ошибочно указано memset(&c, sizeof(c), 0);. См. https://www.rfc-editor.org/errata/eid3608. *Прим. перев.*

```
        ipaddr  dstaddr,          /* IP-адрес получателя */
        int     version,         /* Версия */
        int     mode,            /* Режим хоста */
        int     keyid,           /* Идентификатор ключа */
        int     flags            /* Флаги партнёра */
        )
{
        struct p *p;             /* Указатель на процесс партнёра */

        /*
         * Выделение и инициализация памяти для ассоциации.
         */
        p = malloc(sizeof(struct p));
        p->srcaddr = srcaddr;
        p->dstaddr = dstaddr;
        p->version = version;
        p->hmode = mode;
        p->keyid = keyid;
        p->hpoll = MINPOLL;
        clear(p, X_INIT);
        p->flags = flags;
        return (p);
}

/*
 * find_assoc() - найти соответствующую ассоциацию.
 */
struct p                         /* Указатель на структуру партнёра или NULL */
*find_assoc(
        struct r *r              /* Указатель на принятый пакет */
        )
{
        struct p *p;             /* Указатель на «фиктивную» (dummy) структуру партнёра */

        /*
         * Таблица поиска ассоциаций для сопоставления адреса и порта
         * отправителя, а также режима.
         */
        while (/* все ассоциации */ 0) {
                if (r->srcaddr == p->srcaddr && r->mode == p->hmode)
                        return(p);
        }
        return (NULL);
}

/*
 * md5() - расчёт дайджеста сообщения.
 */
digest
md5(
       int     keyid             /* идентификатор ключа */
       )
{
       /*
        * Расчёт криптографического дайджеста сообщения с ключом.
        * Идентификатор связывается с ключом в локальном кэше ключей. Ключ
        * помещается перед заголовком пакета и полями расширения, результат
        * хэшируется по алгоритму MD5, как указано в RFC 1321. Возвращается
        * код MAC, состоящий из 32-битового идентификатора ключа,
        * объединённого (конкатенация) со 128-битовым дайджестом сообщения.
        */
       return (/* дайджест MD5 */ 0);
}
```

A.3. Интерфейс ввода-вывода ядра

```
   /*
    * Интерфейс с ядром для передачи и приёма пакетов, детали которого
    * зависят от операционной системы.
    *
    * recv_packet — получение пакета из сети.
    */
   struct r                        /* указатель на принятый пакет */
   *recv_packet() {
           return (/* принятый пакет r */ 0);
   }


   /*
    * xmit_packet — передача пакета в сеть.
    */
   void
   xmit_packet(
           struct x *x             /* указатель на передаваемый пакет */
           )
   {
```

```
            /* передача пакета x */
    }
```

A.4. Интерфейс системных часов ядра

```
/*
 * Утилиты системных часов
 *
 * Имеется 3 формата времени: native (Unix), NTP, floating double.
 * Функция get_time() возвращает время в формате NTP. Программы
 * Unix ожидают аргументов в виде структуры из двух 32-битовых слов
 * со знаком в секундах и микросекундах (timeval) или наносекундах
 * (timespec). Функции step_time() и adjust_time() ожидают аргументов
 * floating double со знаком. Упрощённый код здесь показан лишь для
 * иллюстрации и не проверялся.
 */
#define JAN_1970        2208988800UL /* 1970 - 1900 в секундах */

/*
 * get_time — чтение системного времени и преобразование в формат NTP.
 */
tstamp
get_time()
{
        struct timeval unix_time;

        /*
         * Функция вызывается в программе лишь дважды - один раз по
         * прибытии пакета из сети, другой - при включении в очередь на
         * передачу. Вызывает функцию ядра для времени суток (такую как
         * gettimeofday()) и преобразует значение в формат NTP.
         */
        gettimeofday(&unix_time, NULL);
        return (U2LFP(unix_time));
}

/*
 * step_time() - сдвиг системного времени на указанное смещение.
 */
void
step_time(
        double  offset          /* смещение часов */
        )
{
        struct timeval unix_time;
        tstamp  ntp_time;

        /*
         * Преобразование из формата double в native (со знаком) и
         * добавление к текущему времени. Отметим, что добавление
         * происходит в естественном формате, чтобы избежать
         * переполнения и потери точности.
         */
        gettimeofday(&unix_time, NULL);
        ntp_time = D2LFP(offset) + U2LFP(unix_time);
        unix_time.tv_sec = ntp_time >> 32;
        unix_time.tv_usec = (long)(((ntp_time - unix_time.tv_sec) <<
            32) / FRAC * 1e6);
        settimeofday(&unix_time, NULL);
}


/*
 * adjust_time() - сдвиг системных часов на указанное смещение.
 */
void
adjust_time(
        double  offset          /* смещение часов */
        )
{
        struct timeval unix_time;
        tstamp  ntp_time;

        /*
         * Преобразование из формата double в native (со знаком) и
         * добавление к текущему времени.
         */
        ntp_time = D2LFP(offset);
        unix_time.tv_sec = ntp_time >> 32;
        unix_time.tv_usec = (long)(((ntp_time - unix_time.tv_sec) <<
            32) / FRAC * 1e6);
        adjtime(&unix_time, NULL);
}
```

A.5. Партнёрский процесс

```
    /*
```

```
     * Пакет crypto-NAK включает заголовок NTP, за которым следует код MAC,
     * состоящий лишь из идентификатора ключа со значением 0. Это говорит
     * получателю, что предыдущий запрос не был должным образом
     * аутентифицирован, но поля заголовка NTP корректны.
     *
     * Пакет kiss-o'-death содержит заголовок NTP с leap 0x3 (NOSYNC) и
     * stratum 16 (MAXSTRAT). Он говорит получателю, что произошло нечто
     * серьёзное, о чем говорит код kiss в поле refid. Другие поля
     * заголовка NTP могут быть некорректными.
     */
    /*
     * Параметры и константы процесса партнёра.
     */
    #define SGATE           3       /* spike gate (фильтр часов) */
    #define BDELAY          .004    /* широковещательная задержка (s) */

    /*
     * Dispatch codes
     */
    #define ERR             -1      /* Ошибка */
    #define DSCRD           0       /* Отбросить пакет */
    #define PROC            1       /* Обработать пакет */
    #define BCST            2       /* Широковещательный пакет */
    #define FXMIT           3       /* Пакет клиента */
    #define MANY            4       /* Пакет manycast */
    #define NEWPS           5       /* Новый симметричный пассивный клиент */
    #define NEWBC           6       /* Новый широковещательный клиент */

    /*
     * Матрица диспетчеризации
     *              active  passv  client server bcast */
    int table[7][5] = {
    /* nopeer  */   { NEWPS, DSCRD, FXMIT, MANY, NEWBC },
    /* active  */   { PROC,  PROC,  DSCRD, DSCRD, DSCRD },
    /* passv   */   { PROC,  ERR,   DSCRD, DSCRD, DSCRD },
    /* client  */   { DSCRD, DSCRD, DSCRD, PROC,  DSCRD },
    /* server  */   { DSCRD, DSCRD, DSCRD, DSCRD, DSCRD },
    /* bcast   */   { DSCRD, DSCRD, DSCRD, DSCRD, DSCRD },
    /* bclient */   { DSCRD, DSCRD, DSCRD, DSCRD, PROC}
    };

    /*
     * Прочие макросы
     *
     * Этот макрос определяет статус аутентификации. Значение x = 0 говорит
     * о  её необязательности, в иных случаях аутентификация требуется.
     */
    #define AUTH(x, y)      ((x) ? (y) == A_OK : (y) == A_OK || \
                                (y) == A_NONE)

    /*
     * Макросы для функции clear()
     */
    #define BEGIN_CLEAR(p)  ((char *)&((p)->begin_clear))
    #define END_CLEAR(p)    ((char *)&((p)->end_clear))
    #define LEN_CLEAR       (END_CLEAR((struct p *)0) - \
                                BEGIN_CLEAR((struct p *)0))
```

A.5.1. receive()

```
/*
 * receive() - приём пакета и декодирование режима.
 */
void
receive(
        struct r *r             /* Указатель на полученный пакет */
        )
{
        struct p *p;            /* Указатель на структуру партнёра */
        int     auth;           /* Код аутентификации */
        int     has_mac;        /* Размер MAC */
        int     synch;          /* Состояние синхронизации */

        /*
         * Проверяет списки контроля доступа, служащего для поддержки
         * списка разрешённых или запрещённых адресов IP. Могут быть
         * разные списки для аутентифицированных и неаутентифицированных
         * клиентов.
         */
        if (!access(r))
                return;                 /* Доступ отклонён */

        /*
         * Версия не должна быть в будущем. Проверка формата включает
         * размера пакета, MAC и полей заголовка при их наличии.
         */
        if (r->version > VERSION /* или ошибка формата */)
```

```
                return;                 /* ошибка формата */

        /*
         * Условие аутентификации задаётся двумя параметрами на уровне
         * клиента
         * P_NOPEER     не активировать ассоциацию до аутентификации.
         * P_NOTRUST    не разрешать доступ  до аутентификации
         *              (предполагает P_NOPEER).
         *
         * Выходные значения
         * A_NONE       пакет не включает MAC.
         * A_OK         пакет содержит MAC и аутентификация успешна.
         * A_ERROR      пакет имеет MAC, но проверка не прошла.
         * A_CRYPTO     crypto-NAK. MAC содержит лишь 4 октета.
         *
         * Примечание. Макрос AUTH (x, y) служит для фильтрации вывода.
         * При x=0 приемлемыми результатами будут NONE и OK, при x=1 - OK.
         */

        has_mac = /* размер поля MAC */ 0;
        if (has_mac == 0) {
                auth = A_NONE;          /* не требуется */
        } else if (has_mac == 4) {
                auth = A_CRYPTO;        /* crypto-NAK */
        } else {
                if (r->mac != md5(r->keyid))
                        auth = A_ERROR; /* ошибка аутентификации */
                else
                        auth = A_OK;    /* проверка прошла */
        }

        /*
         * Поиск ассоциации и диспетчеризация кода. Если ассоциации не
         * найдено, предполагается p->hmode = NULL.
         */
        p = find_assoc(r);
        switch(table[(unsigned int)(p->hmode)][(unsigned int)(r->mode)])
        {

        /*
         * Пакет клиента без ассоциации. Передача отклика сервера без
         * сохранения состояния.
         */
        case FXMIT:

                /*
                 * Для unicast-адреса получателя передача пакета сервера.
                 * При отказе аутентификации передача пакета crypto-NAK.
                 */

                /* Адрес получателя не групповой */[1]
                if (0) {
                        if (AUTH(p->flags & P_NOTRUST, auth))
                                fast_xmit(r, M_SERV, auth);
                        else if (auth == A_ERROR)
                                fast_xmit(r, M_SERV, A_CRYPTO);
                        return;         /* M_SERV packet sent */
                }

                /*
                 * Должно быть manycast. Отклик не передаётся, если нет
                 * синхронизации или stratum больше чем у manycast.
                 */
                if (s.leap == NOSYNC || s.stratum > r->stratum)
                        return;

                /*
                 * Отклик лишь при успешной аутентификации. Применяется
                 * адрес unicast, а не multicast.
                 */
                if (AUTH(p->flags & P_NOTRUST, auth))
                        fast_xmit(r, M_SERV, auth);
                return;

        /*
         * Новая временная ассоциация клиента manycast. Активизируется с
         * указанной в пакете версией. При отказе аутентификации пакет
         * игнорируется. Серверный пакет проверяется сравнением метки
         * r->org в пакете с меткой p->xmt в групповой клиентской
         * ассоциации. При совпадении пакет сервера аутентичен.
         */

        case MANY:
                if (!AUTH(p->flags & (P_NOTRUST | P_NOPEER), auth))
```

[1]Ошибка, см. https://www.rfc-editor.org/errata/eid2514. *Прим. перев.*

```
                return;         /* Ошибка аутентификации */

        p = mobilize(r->srcaddr, r->dstaddr, r->version, M_CLNT,
            r->keyid, P_EPHEM);
        break;

/*
 * Новая симметричная пассивная ассоциация.  Активизируется с
 * указанной в пакете версией. При отказе аутентификации передаётся
 * пакет crypto-NAK. Если активация не разрешена, передаётся
 * пакет симметричный активной ассоциации.
 */
case NEWPS:
        if (!AUTH(p->flags & P_NOTRUST, auth)) {
                if (auth == A_ERROR)
                        fast_xmit(r, M_SACT, A_CRYPTO);
                return;         /* передача crypto-NAK */
        }
        if (!AUTH(p->flags & P_NOPEER, auth)) {
                fast_xmit(r, M_SACT, auth);
                return;         /* передача M_SACT */
        }
        p = mobilize(r->srcaddr, r->dstaddr, r->version, M_PASV,
            r->keyid, P_EPHEM);
        break;

/*
 * Новая широковещательная клиентская ассоциация. Активизируется с
 * указанной в пакете версией. При отказе аутентификации пакет
 * игнорируется. Отметим, что этот код не поддерживает начальный
 * блок передач (volley) в эталонной реализации.
 */
case NEWBC:
        if (!AUTH(p->flags & (P_NOTRUST | P_NOPEER), auth))
                return;         /* Ошибка аутентификации */

        if (!(s.flags & S_BCSTENAB))
                return;         /* Широковещание не разрешено */

        p = mobilize(r->srcaddr, r->dstaddr, r->version, M_BCLN,
            r->keyid, P_EPHEM);
        break;                  /* Обработка продолжается */

/*
 * Обработка пакета (заглушка).
 */
case PROC:
        break;                  /* Обработка продолжается */

/*
 * Недействительная комбинация режимов. Это возникает лишь
 * для временных ассоциаций, поэтому корректно просто бросить.
 */
case ERR:
        clear(p, X_ERROR);
        return;          /* Недействительная комбинация режимов */

/*
 * Нет совпадения, пакет просто отбрасывается.
 */
case DSCRD:
        return;                 /* Брошенный сирота */
}

/*
 * Далее выполняется строгое планирование проверки меток. Если
 * метка передачи имеет значение 0, сервер просто повреждён.
 */
if (r->xmt == 0)
        return;                 /* Недействительная метка */

/*
 * Если метка дублирует предыдущую, пакет использован повторно.
 */
if (r->xmt == p->xmt)
        return;                 /* Дубликат пакета */

/*
 * Если это пакет широковещательного режима, проверка завершается.
 * Если метка источника 0, отправитель ещё не слышал нас. Иначе,
 * если метка не совпадает с меткой передачи, пакет фальшивый.
 */
synch = TRUE;
if (r->mode != M_BCST) {
        if (r->org == 0)
                synch = FALSE;  /* Не синхронизировано */
```

```
                else if (r->org != p->xmt)
                        synch = FALSE;  /* фиктивный пакет */
        }

        /*
         * Обновление меток источника и получателя. Если нет
         * синхронизации или пакет фиктивный, доставка прерывается.
         */
        p->org = r->xmt;
        p->rec = r->dst;
        if (!synch)
                return;                 /* нет синхронизации */

        /*
         * Метка действительная и принятый пакет соответствует последнему
         * переданному. Пакет crypto-NAK может говорить о смене ключей
         * сервером. Ассоциация деактивируется до лучших времён.
         */
        if (auth == A_CRYPTO) {
                clear(p, X_CRYPTO);
                return;                 /* crypto-NAK */
        }

        /*
         * Если ассоциация аутентифицирована, key ID отличен от 0 и
         * принятый должен аутентифицироваться. Это предназначено для
         * предотвращения атак bait-and-switch (заманить и подменить),
         * возможных в прежних версиях.
         */
        if (!AUTH(p->keyid || (p->flags & P_NOTRUST), auth))
                return;                 /* bad auth */

        /*
         * Сделано всё возможное для проверки меток времени и
         * предотвращения возможности нарушить работу протокола или
         * внедрить фиктивные данные.
         */
        packet(p, r);
}
```

A.5.1.1. packet()

```
/*
 * packet() - обработка пакета и расчёт смещения, задержки и дисперсии.
 */
void
packet(
        struct p *p,            /* Указатель на структуру партнёра */
        struct r *r             /* Указатель на полученный пакет */
        )
{
        double  offset;         /* Смещение выборки */
        double  delay;          /* Задержка выборки */
        double  disp;           /* Дисперсия выборки */

        /*
         * Пакет действителен, рассматриваются остальные поля заголовка.
         * Отметим, что stratum 0 (не задано) отображается на MAXSTRAT
         * для упрощения сравнения слоёв и обеспечения естественного
         * интерфейса с драйверами радиочасов, работающими со stratum 0.
         */
        p->leap = r->leap;
        if (r->stratum == 0)
                p->stratum = MAXSTRAT;
        else
                p->stratum = r->stratum;
        p->pmode = r->mode;
        p->ppoll = r->poll;
        p->rootdelay = FP2D(r->rootdelay);
        p->rootdisp = FP2D(r->rootdisp);
        p->refid = r->refid;
        p->reftime = r->reftime;

        /*
         * Проверка синхронизации сервера с действительным слоем и того,
         * что время эталона не позднее времени передачи.
         */
        if (p->leap == NOSYNC || p->stratum >= MAXSTRAT)
                return;                 /* нет синхронизации */

        /*
         * Проверка пригодности корневой дистанции.[1]
         */
        if (p->rootdelay / 2 + p->rootdisp >= MAXDISP || p->reftime >
            r->xmt)
```

[1] В оригинале приведённое ниже условие содержало ошибки. https://www.rfc-editor.org/errata/eid5978. *Прим. перев.*

```
                return;                 /* Непригодные значения заголовка */

        poll_update(p, p->hpoll);
        p->reach |= 1;

        /*
         * Расчёт смещения, задержки и дисперсии и передача их фильтру
         * часов. Отметим подразумеваемую обработку. Разность первого
         * порядка определяется напрямую в 64-битовой арифметике и
         * результат преобразуется в floating double. Дальнейшая обработка
         * использует арифметику floating double с аппаратным округлением.
         * Это нужно для предотвращения переполнений и потери точности.
         *
         * Расчёт задержки является особым случаем. Если часы сервера и
         * клиента работают с разной скоростью, а сеть очень быстрая,
         * задержка может стать отрицательной. Для предотвращения проблем
         * задержка должна быть не меньше точности системы[1].
         */
        if (p->pmode == M_BCST) {
                offset = LFP2D(r->xmt - r->dst);
                delay = BDELAY;
                disp = LOG2D(r->precision) + LOG2D(s.precision) + PHI *
                    2 * BDELAY;
        } else {
                offset = (LFP2D(r->rec - r->org) + LFP2D(r->xmt -
                    r->dst)) / 2;
                delay = max(LFP2D(r->dst - r->org) - LFP2D(r->xmt -
                    r->rec), LOG2D(s.precision));
                disp = LOG2D(r->precision) + LOG2D(s.precision) + PHI *
                    LFP2D(r->dst - r->org);
        }
        clock_filter(p, offset, delay, disp);
}
```

A.5.2. clock_filter()

```
/*
 * clock_filter(p, offset, delay, dispersion) — выбор лучшей из 8 последних
 * выборок задержка/смещения.
 */
void
clock_filter(
        struct p *p,            /* Указатель на структуру партнёра */
        double  offset,         /* Смещение часов */
        double  delay,          /* Круговая задержка */
        double  disp            /* Дисперсия */
        )
{
        struct f f[NSTAGE];     /* Сортированный список */
        double  dtemp;
        int     i;

        /*
         * Фильтр часов включает 8 квартетов (offset, delay, dispersion,
         * time). Выполняется сдвиг на 1 квартет влево с отбрасыванием
         * левого квартета. При сдвиге каждого квартета дисперсия растёт
         * поскольку обновляется последний фильтр. Одновременно каждый
         * квартет копируется во временный список, а потом offset, delay,
         * disp, time) помещается в пустую позицию справа.
         */
        for (i = 1; i < NSTAGE; i++) {
                p->f[i] = p->f[i - 1];
                p->f[i].disp += PHI * (c.t - p->t);
                f[i] = p->f[i];
        }
        p->f[0].t = c.t;
        p->f[0].offset = offset;
        p->f[0].delay = delay;
        p->f[0].disp = disp;
        f[0] = p->f[0];

        /*
         * Сортировка временного списка по росту f[].delay. Первая запись
         * сортированного списка указывает лучшую выборку, но может быть
         * старой.
         */
        dtemp = p->offset;
        p->offset = f[0].offset;
        p->delay = f[0].delay;
        for (i = 0; i < NSTAGE; i++) {
                p->disp += f[i].disp / (1 << (i + 1));[2]
                p->jitter += SQUARE(f[i].offset - f[0].offset);
        }
        p->jitter = max(SQRT(p->jitter), LOG2D(s.precision));
```

[1] В оригинале приведённый ниже код содержал ошибки. См. https://www.rfc-editor.org/errata/eid3125. *Прим. перев.*

[2] В оригинале эта строка содержит ошибку. См. https://www.rfc-editor.org/errata/eid6550. *Прим. перев.*

```
        /*
         * Применять выборку лишь один раз и никогда не применять выборку
         * старше последней, но сделанной до синхронизации.
         */
        if (f[0].t - p->t <= 0 && s.leap != NOSYNC)
                return;

        /*
         * Сравниваются разность последнего и текущего смещения с текущим
         * значением jitter). Если разность больше SGATE (3) и интервал с
         * момента получения последнего смещения меньше удвоенного
         * интервала опроса, всплеск отбрасывается. Иначе, если не в режиме
         * burst, выбираются истинные часы.
         */
        if (fabs(p->offset - dtemp) > SGATE * p->jitter && (f[0].t -
            p->t) < 2 * s.poll)
                return;

        p->t = f[0].t;
        if (p->burst == 0)
                clock_select();
        return;
}

/*
 * fit() - проверка пригодности ассоциации p для синхронизации.
 */
int
fit(
        struct p *p             /* Указатель на структуру партнёра */
        )
{
        /*
         * Ошибка stratum, если (1) сервер не был синхронизирован,
         * (2) слой сервера недействителен.
         */
        if (p->leap == NOSYNC || p->stratum >= MAXSTRAT)
                return (FALSE);

        /*
         * Ошибка дистанции возникает, если корневая дистанция
         * превышает порог плюс один интервал опроса.
         */
        if (root_dist(p) > MAXDIST + PHI * LOG2D(s.poll))
                return (FALSE);

        /*
         * Петля (ошибка), если удалённый партнёр синхронизирован с
         * локальным или с текущим партнёром системы. Отметим, что это
         * относится к IPv4, а для IPv6 применяется хэш MD5.
         */
        if (p->refid == p->dstaddr || p->refid == s.refid)
                return (FALSE);

        /*
         * Ошибка доступности возникает, если сервер недоступен.
         */
        if (p->reach == 0)
                return (FALSE);

        return (TRUE);
}

/*
 * clear() - повторная инициализация ассоциации, деактивация
 * временной ассоциации.
 */
void
clear(
        struct p *p,            /* Указатель на структуру партнёра */
        int     kiss            /* kiss-код */
        )
{
        int i;

        /*
         * Сначала возвращаются все ресурсы. Типовые ресурсы не указаны
         * здесь, но включают динамически созданные структуры для ключей,
         * сертификатов и т. п. Если ассоциация временная и нет
         * повторной инициализации, просто возвращается память ассоциации.
         */
        /* Возврат ресурсов */
        if (s.p == p)
                s.p = NULL;
        if (kiss != X_INIT && (p->flags & P_EPHEM)) {
```

```
                free(p);
                return;
        }

        /*
         * Инициализация полей ассоциации для общего сброса.
         */
        memset(BEGIN_CLEAR(p), LEN_CLEAR, 0);
        p->leap = NOSYNC;
        p->stratum = MAXSTRAT;
        p->ppoll = MAXPOLL;
        p->hpoll = MINPOLL;
        p->disp = MAXDISP;
        p->jitter = LOG2D(s.precision);
        p->refid = kiss;
        for (i = 0; i < NSTAGE; i++)
                p->f[i].disp = MAXDISP;

        /*
         * Первый опрос делается случайным на случай одновременного
         * возникновения тысяч широковещательных клиентов после
         * долгого отсутствия широковещательного сервера.
         */
        p->outdate = p->t = c.t;
        p->nextdate = p->outdate + (random() & ((1 << MINPOLL) - 1));
}
```

A.5.3. fast_xmit()

```
/*
 * fast_xmit() - передача отклика на принятый пакет r
 */
void
fast_xmit(
        struct r *r,            /* Указатель на полученный пакет */
        int     mode,           /* Режим ассоциации */
        int     auth            /* Код аутентификации */
        )
{
        struct x x;

        /*
         * Инициализация заголовка и метки времени передачи. Отметим, что
         * поле version копируется из принятого пакета для совместимости
         * с прежними версиями.
         */
        x.version = r->version;
        x.srcaddr = r->dstaddr;
        x.dstaddr = r->srcaddr;
        x.leap = s.leap;
        x.mode = mode;
        if (s.stratum == MAXSTRAT)
                x.stratum = 0;
        else
                x.stratum = s.stratum;
        x.poll = r->poll;
        x.precision = s.precision;
        x.rootdelay = D2FP(s.rootdelay);
        x.rootdisp = D2FP(s.rootdisp);
        x.refid = s.refid;
        x.reftime = s.reftime;
        x.org = r->xmt;
        x.rec = r->dst;
        x.xmt = get_time();

        /*
         * Если код аутентификации имеет значение A.NONE, включается лишь
         * заголовок, при A.CRYPTO передаётся crypto-NAK, для A.OK -
         * действительный код MAC. Используется key ID из принятого пакета
         * и ключ из локального кэша.
         */
        if (auth != A_NONE) {
                if (auth == A_CRYPTO) {
                        x.keyid = 0;
                } else {
                        x.keyid = r->keyid;
                        x.dgst = md5(x.keyid);
                }
        }
        xmit_packet(&x);
}
```

A.5.4. access()

```
 /*
  * access() - задаёт ограничения при доступе.
  */
```

```
 int
 access(
         struct r *r             /* Указатель на полученный пакет */
         )
 {
         /*
          * Список контроля доступа — это неупорядоченный набор кортежей
          * из адреса, маски и слова ограничения, содержащего определённые
          * биты. В списке отыскивается первое совпадение по адресу
          * отправителя (r->srcaddr) и возвращаются заданные ограничения.
          */
         return (/* биты доступа */ 0);
 }
```

A.5.5. Системный процесс

A.5.5.1. clock_select()

```
/*
 * clock_select() - поиск лучших часов
 */
void
clock_select() {
       struct p *p, *osys;     /* Указатели на структуры партнёров */
       double  low, high;      /* Протяжённость интервалов корректности */
       int     allow, found, chime; /* Применяется алгоритмом пересечения */
       int     n, i, j;

        /*
         * Сначала исключаются из набора серверов ложные часы и остаются
         * лишь истинные. Интервалом корректности для ассоциации p служит
         * интервал от offset - root_dist() до offset + root_dist().
         * Задача состоит в отборе кандидатов, т. е. поиске пересечения
         * интервалов корректности числом более половины набора серверов.
         *
         * Сначала создаётся список триплетов (p, type, edge), как показано
         * ниже, затем список сортируется по росту edge.[1]
         */
        osys = s.p;
        s.p = NULL;
        n = 0;
        while (fit(p)) {
                s.m[n].p = p;
                s.m[n].type = +1;
                s.m[n].edge = p->offset + root_dist(p);
                n++;
                s.m[n].p = p;
                s.m[n].type = 0;
                s.m[n].edge = p->offset;
                n++;
                s.m[n].p = p;
                s.m[n].type = -1;
                s.m[n].edge = p->offset - root_dist(p);
                n++;
        }

        /*
         * Поиск наибольшего непрерывного пересечения интервалов
         * корректности. Значение allow указывает число разрешённых
         * ложных часов, found - число найденных midpoint. Отметим, что
         * значения edge ограничены диапазоном +-(2 ^ 30) < +-2e9
         * расчётами меток времени.
         */
        low = 2e9; high = -2e9;
        for (allow = 0; 2 * allow < n; allow++) {

                /*
                 * Сканирование списка кандидатов от низшего в высшему
                 * для поиска низшей конечной точки.[2]
                 */
                found = 0;
                chime = 0;
                for (i = 0; i < n; i++) {
                        chime -= s.m[i].type;
                        if (chime >= n - allow) {
                                low = s.m[i].edge;
                                break;
                        }
                        if (s.m[i].type == 0)
                                found++;
                }

                /*
```

[1] См. https://www.rfc-editor.org/errata/eid6280. *Прим. перев.*

[2] В оригинале циклы for содержали ошибки. См. https://www.rfc-editor.org/errata/eid6207 и https://www.rfc-editor.org/errata/eid3127. *Прим. перев.*

```
		 * Сканирование списка кандидатов от высшего к низшему
		 * для поиска высшей конечной точки.
		 */
		chime = 0;
		for (i = n - 1; i >= 0; i--) {
			chime += s.m[i].type;
			if (chime >= n - allow) {
				high = s.m[i].edge;
				break;
			}
			if (s.m[i].type == 0)
				found++;
		}

		/*
		 * Если число midpoint больше числа разрешённых ложных
		 * часов, пересечение включает хотя бы одни истинные часы
		 * без midpoint. В этом случае инкрементируется число
		 * разрешённых ложных часов и процесс повторяется. Если это
		 * не так и пересечение не пусто, заявляется успех.
		 */
		if (found > allow)
			continue;

		if (high > low)
			break;
	}

	/*
	 * Алгоритм кластеризации, создающий список оставшихся пар (p,
	 * metric) из списка кандидатов, где metric определяется первым
	 * по stratum, затем по корневой дистанции. При прочих равных это
	 * задаёт порядок предпочтения.
	 */
	s.n = 0;
	for (i = 0; i < n; i++) {
		if (s.m[i].edge < low || s.m[i].edge > high)
			continue;

		p = s.m[i].p;
		s.v[n].p = p;
		s.v[n].metric = MAXDIST * p->stratum + root_dist(p);
		s.n++;
	}

	/*
	 * Для соблюдения утверждений о корректности должно остаться
	 * хотя бы NSANE кандидатов. Исходно византийский критерий
	 * требует 4 оставшихся, но для демонстрации достаточно 1.
	 */
	if (s.n < NSANE)
		return;

	/*
	 * Для каждой ассоциации p поочередно рассчитывается выбор
	 * jitter p->sjitter как квадратного корня из суммы квадратов
	 * (p->offset - q->offset) для всех q ассоциаций. Идея состоит в
	 * повторяющемся отбрасывании оставшихся с максимальной вариацией,
	 * пока не будут выполнены условия завершения.
	 */
	while (1) {
		struct p *p, *q, *qmax; /* Указатели на структуры партнёров */
		double  max, min, dtemp;

		max = -2e9; min = 2e9;
		for (i = 0; i < s.n; i++) {
			p = s.v[i].p;
			if (p->jitter < min)
				min = p->jitter;
			dtemp = 0;
			for (j = 0; j < n; j++) {
				q = s.v[j].p;
				dtemp += SQUARE(p->offset - q->offset);
			}
			dtemp = SQRT(dtemp);
			if (dtemp > max) {
				max = dtemp;
				qmax = q;
			}
		}

		/*
		 * Если максимальная вариация выбора меньше минимальной
		 * вариации партнёра, дополнительное отбрасывание не
		 * повысит минимальную вариацию партнёра, поэтому можно
		 * остановиться. Для уверенности в сохранении достаточного
```

```
                 * для алгоритма кластеризации числа оставшихся процесс
                 * завершается, если их число не превышает NMIN (3).
                 */
                if (max < min || n <= NMIN)
                        break;

                /*
                 * Удаление оставшегося с qmax и повтор процесса.
                 */
                s.n--;
        }

        /*
         * Выбор лучших часов. Если прежний партнёр системы имеется в
         * списке с тем же слоем, что и у первого в списке, часы не
         * обновляются. Иначе выбирается первый оставшийся как новый
         * партнёр системы.
         */
        if (osys->stratum == s.v[0].p->stratum)
                s.p = osys;
        else
                s.p = s.v[0].p;
        clock_update(s.p);
}
```

A.5.5.2. root_dist()

```
/*
 * root_dist() - расчёт корневой дистанции.
 */
double
root_dist(
        struct p *p             /* Указатель на структуру партнёра */
        )
{

        /*
         * Корневой дистанцией синхронизации служит максимальная ошибка
         * из-за всех причин локальных часов относительно первичного
         * сервера. Она определяется как половина суммы общей задержки,
         * общей дисперсии и вариации партнера.
         */
        return (max(MINDISP, p->rootdelay + p->delay) / 2 +
            p->rootdisp + p->disp + PHI * (c.t - p->t) + p->jitter);
}
```

A.5.5.3. accept()

```
/*
 * accept() - проверка пригодности ассоциации p для синхронизации.
 */
int
accept(
        struct p *p             /* Указатель на структуру партнёра */
        )
{
        /*
         * Ошибка stratum возникает, если (1) сервер не синхронизирован,
         * (2) слой сервера не действителен.
         */
        if (p->leap == NOSYNC || p->stratum >= MAXSTRAT)
                return (FALSE);

        /*
         * Ошибка дистанции возникает, если корневая дистанция
         * превышает порог плюс один интервал опроса.
         */
        if (root_dist(p) > MAXDIST + PHI * LOG2D(s.poll))
                return (FALSE);

        /*
         * Петля (ошибка), если удалённый партнёр синхронизирован с
         * локальным или с текущим партнёром системы. Отметим, что это
         * относится к IPv4, а для IPv6 применяется хэш MD5.
         */
        if (p->refid == p->dstaddr || p->refid == s.refid)
                return (FALSE);

        /*
         * Ошибка доступности возникает, если сервер недоступен.
         */
        if (p->reach == 0)
                return (FALSE);

        return (TRUE);
}
```

A.5.5.4. clock_update()

```
/*
 * clock_update() - обновление системных часов.
 */
void
clock_update(
        struct p *p             /* Указатель на структуру партнёра */
        )
{
        double dtemp;

        /*
         * Старое обновление, например, в результате смены партнёра системы
         * не выполняется. Старые и совпадающие выборки не применяются.
         */
        if (s.t >= p->t)
                return;

        /*
         * Объединение оставшихся смещения и обновление системных часов,
         * функция local_clock() будет указывать результат.
         */
        s.t = p->t;
        clock_combine();
        switch (local_clock(p, s.offset)) {

        /*
         * Смещение слишком велико и может быть фиктивным. Это вносится в
         * журнал системы и оператор оповещается для установки часов
         * вручную в диапазоне PANIC. Эталонная реализация включает опции
         * команды для отключения этой проверки и смены порога паники,
         * принятого по умолчанию (1000 секунд).
         */
        case PANIC:
                exit (0);

        /*
         * Смещение больше шага ступени (0,125 сек по умолчанию). После
         * применения шага все ассоциации будут иметь несогласованные
         * значения времени, поэтому они сбрасываются и запускаются снова.
         * Порог шага можно изменить в эталонной реализации для снижения
         * вероятности того, что часы могут быть переведены обратно.
         * Однако возможны серьёзные последствия, как отмечено в
         * публикации на сайте проекта NTP.
         */
        case STEP:
                while (/* all associations */ 0)
                        clear(p, X_STEP);
                s.stratum = MAXSTRAT;
                s.poll = MINPOLL;
                break;

        /*
         * Смещение меньше порога шага, что является обычным делом.
         * Системные переменные обновляются по переменным партнёра.
         * Нижний предел роста дисперсии служит для предотвращения
         * петель и скачков часов при вступлении в игру прецизионных
         * источников. Предел можно сменить с принятого по умолчанию
         * значения 0,01 сек в эталонной реализации.
         */
        case SLEW:
                s.leap = p->leap;
                s.stratum = p->stratum + 1;
                s.refid = p->refid;
                s.reftime = p->reftime;
                s.rootdelay = p->rootdelay + p->delay;
                dtemp = SQRT(SQUARE(p->jitter) + SQUARE(s.jitter));
                dtemp += max(p->disp + PHI * (c.t - p->t) +
                    fabs(p->offset), MINDISP);
                s.rootdisp = p->rootdisp + dtemp;
                break;

        /*
         * некоторые выборки отбрасываются, хотя, например, прямое
         * измерение частоты происходит.
         */
        case IGNORE:
                break;
        }
}
```

A.5.5.5. clock_combine()

```
/*
 * clock_combine() - объединение смещений.
 */
void
```

```
clock_combine()
{
        struct p *p;            /* Указатель на структуру партнёра */
        double x, y, z, w;
        int     i;

        /*
         * Объединение смещений, оставшихся после алгоритма кластеризации,
         * с использованием взвешенного среднего по весу, определяемому
         * корневой дистанцией. Расчёт вариаций выбора как взвешенная RMS
         * разница между первым и остальными кандидатами. В некоторых
         * случаях наследуемые вариации часов можно снизить, отказавшись от
         * этого алгоритма, особенно при частом переводе часов. Эталонную
         * реализацию можно настроить на отказ от этого алгоритма указанием
         * предпочтительного партнёра.
         */
        y = z = w = 0;
        for (i = 0; s.v[i].p != NULL; i++) {
                p = s.v[i].p;
                x = root_dist(p);
                y += 1 / x;
                z += p->offset / x;
                w += SQUARE(p->offset - s.v[0].p->offset) / x;
        }
        s.offset = z / y;
        s.jitter = SQRT(w / y);
}
```

A.5.5.6. local_clock()

```
/*
 * Параметры и константы дисциплины часов.
 */
#define STEPT           .128    /* Порог шага (s) */
#define WATCH           900     /* Порог ступени (stepout) (s) */
#define PANICT          1000    /* Порог паники (s) */
#define PLL             65536   /* Усиление контура PLL */
#define FLL             MAXPOLL + 1 /* Усиление контура FLL */
#define AVG             4       /* Константа усреднения параметров */
#define ALLAN           1500    /* Компрометация перехвата Allan (s) */
#define LIMIT           30      /* Порог корректировки опроса */
#define MAXFREQ         500e-6  /* Допуск частоты (500 ppm) */
#define PGATE           4       /* Диапазон корректировки опроса */

/*
 * local_clock() - дисциплина локальных часов.
 */
int                             /* код завершения */
local_clock(
        struct p *p,            /* Указатель на структуру партнёра */
        double  offset          /* Смещение часов из combine() */
        )
{
        int     state;          /* Статус дисциплины часов */
        double  freq;           /* Частота */
        double  mu;             /* Интервал с последнего обновления */
        int     rval;
        double  etemp, dtemp;

        /*
         * Отказ и возврат при слишком большом смещении.
         */
        if (fabs(offset) > PANICT)
                return (PANIC);

        /*
         * Функция смены состояния конечного автомата. Именно здесь
         * выполняется действие и определяется реакция системы на
         * слишком большие ошибки времени и частоты. Есть 2 основных
         * режима в зависимости от превышения порога шага (ступени).
         */
        rval = SLEW;
        mu = p->t - s.t;
        freq = 0;
        if (fabs(offset) > STEPT) {
                switch (c.state) {

                /*
                 * В состоянии S_SYNC игнорируется первый всплеск и статус
                 * меняется на S_SPIK.
                 */
                case SYNC:
                        state = SPIK;
                        return (rval);

                /*
                 * В состоянии S_FREQ игнорируются всплески и провалы. При
```

```
                 * первом всплеске после порога stepout рассчитывается
                 * коррекция частоты и шаг времени.
                 */
                case FREQ:
                        if (mu < WATCH)
                                return (IGNORE);

                        freq = (offset - c.offset) / mu;
                        /* переход в S_SPIK */

                /*
                 * В состоянии S_SPIK игнорируются последующие всплески,
                 * пока не будет найден выступ или превышен порог stepout.
                 */
                case SPIK:
                        if (mu < WATCH)
                                return (IGNORE);

                        /* Возврат к принятому по умолчанию */

                /*
                 * По умолчанию сюда попадают в состоянии S_NSET и S_FSET,
                 * в также сверху в состоянии S_FREQ. Применяется шаг
                 * времени и снижается интервал опроса.
                 *
                 * В S_NSET исходная корректировка частоты недоступна
                 * обычно по причине того, что файл частоты ещё не записан.
                 * Поскольку время не попадает в диапазон захвата,
                 * применяется шаг часов. Частота устанавливается напрямую
                 * после интервала stepout.
                 *
                 * В S_FSET исходная корректировка частоты выполняется из
                 * файла. Поскольку время не попадает в диапазон захвата,
                 * часы переводятся незамедлительно, а не по истечении
                 * интервала stepout. Если первый перевод часов занимает
                 * 17 минут, люди начинают нервничать.
                 *
                 * В S_SPIK превышен порог stepout и фаза остаётся выше
                 * порога шага. Отметим, что один пик выше порога шага
                 * всегда подавляется даже при более длинных интервалах
                 * опроса.
                 */
                default:

                        /*
                         * Функция установки времени ядра обычно
                         * через системный вызов Unix settimeofday().
                         */
                        step_time(offset);
                        c.count = 0;
                        s.poll = MINPOLL;
                        rval = STEP;
                        if (state == NSET) {
                                rstclock(FREQ, p->t, 0);
                                return (rval);
                        }
                        break;
                }
                rstclock(SYNC, p->t, 0);
        } else {

                /*
                 * Расчёт вариаций часов как RMS экспоненциально взвешенных
                 * разностей смещения. Применяется кодом настройки опроса.
                 */
                etemp = SQUARE(c.jitter);
                dtemp = SQUARE(max(fabs(offset - c.last),
                    LOG2D(s.precision)));
                c.jitter = SQRT(etemp + (dtemp - etemp) / AVG);
                switch (c.state) {

                /*
                 * В S_NSET это первое полученное обновление и частота ещё
                 * не инициализирована. Первым делом напрямую измеряется
                 * частота тактового генератора.
                 */
                case NSET:
                        rstclock(FREQ, p->t, offset);
                        return (IGNORE);

                /*
                 * В S_FSET это первое полученное обновление и частота ещё
                 * не инициализирована. Настраивается фаза, но частота не
                 * настраивается до следующего обновления.
                 */
                case FSET:
```

```
                rstclock(SYNC, p->t, offset);
                break;

        /*
         * В S_FREQ обновления игнорируются до достижения порога
         * stepout. После этого корректировка фазы и частоты с
         * переходом в S_SYNC.
         */
        case FREQ:
                if (c.t - s.t < WATCH)
                        return (IGNORE);

                freq = (offset - c.offset) / mu;
                break;

        /*
         * Сюдя по умолчанию попадают в состоянии S_SYNC и S_SPIK.
         * Рассчитывается обновление частоты по данным PLL и FLL.
         */
        default:

                /*
                 * Константы FLL и PLL для изменения частоты
                 * зависят от интервала опроса и перехвата Allan
                 * FLL не применяется ниже половины перехвата
                 * Allan. Выше него усиление контура растёт с
                 * шагом 1 / AVG.
                 */
                if (LOG2D(s.poll) > ALLAN / 2) {
                        etemp = FLL - s.poll;
                        if (etemp < AVG)
                                etemp = AVG;
                        freq += (offset - c.offset) / (max(mu,
                            ALLAN) * etemp);
                }

                /*
                 * Для PLL интервалом интеграции (numerator)
                 * служит минимальное из значений интервалов
                 * обновления и опроса. Это разрешает более частые
                 * избыток выборок, но не их недостаток.
                 */
                etemp = min(mu, LOG2D(s.poll));
                dtemp = 4 * PLL * LOG2D(s.poll);
                freq += offset * etemp / (dtemp * dtemp);
                rstclock(SYNC, p->t, offset);
                break;
        }
}

/*
 * Расчёт новой частоты и её стабильности (дрейф). Расчёт дрейфа
 * часов как RMS экспоненциально взвешенных разностей частот. Это
 * не применяется напрямую, но вместе с вариациями может быть очень
 * полезно для мониторинга и отладки.
 */
freq += c.freq;
c.freq = max(min(MAXFREQ, freq), -MAXFREQ);
etemp = SQUARE(c.wander);
dtemp = SQUARE(freq);
c.wander = SQRT(etemp + (dtemp - etemp) / AVG);

/*
 * Настраивается интервал опроса путём сравнения текущего смещения
 * с вариациями часов. Если смещение меньше, значения вариаций
 * часов, умноженного на константу, средний интервал увеличивается,
 * иначе снижается. Небольшой гистерезис повысит стабильность.
 * лучше всего это работает в режиме burst.
 */
if (fabs(c.offset) < PGATE * c.jitter) {
        c.count += s.poll;
        if (c.count > LIMIT) {
                c.count = LIMIT;
                if (s.poll < MAXPOLL) {
                        c.count = 0;
                        s.poll++;
                }
        }
} else {
        c.count -= s.poll << 1;
        if (c.count < -LIMIT) {
                c.count = -LIMIT;
                if (s.poll > MINPOLL) {
                        c.count = 0;
                        s.poll--;
                }
```

```
                }
        }
        return (rval);
}
```

A.5.5.7. rstclock()

```
  /*
   * rstclock() - конечный автомат часов.
   */
  void
  rstclock(
          int     state,          /* Новое состояние */
          double  offset,         /* Новое смещение */
          double  t               /* Новое время обновления */
          )
  {
          /*
           * Переход в новое состояние и установка переменных состояния.
           * Отметим, что применяется время последней выборки фильтра
           * часов, которое должно быть раньше текущего времени.
           */
          c.state = state;
          c.last = c.offset = offset;
          s.t = t;
  }
```

A.5.6. Процесс корректировки часов

A.5.6.1. clock_adjust()

```
 /*
  * clock_adjust() - запускается с интервалом в 1 секунду.
  */
 void
 clock_adjust() {
          double  dtemp;

          /*
           * Обновление времени процесса c.t и увеличение дисперсии с
           * момента последнего обновления. В отличие от NTPv3, NTPv4 не
           * заявляет отсутствие синхронизации по истечении суток,
           * поскольку для этого служит порог дисперсии. Когда дисперсия
           * превышает MAXDIST (1 s), сервер считается неподходящим
           * для синхронизации.
           */
          c.t++;
          s.rootdisp += PHI;

          /*
           * Настройка фазы и частоты. Коэффициент усиления (denominator)
           * не разрешается увеличивать сверх перехвата Allan. Не имеет
           * смысла усреднять фазовый шум за пределами этой точки и это
           * помогает демпфировать остаточное смещение при больших
           * интервалах опроса.
           */
          dtemp = c.offset / (PLL * min(LOG2D(s.poll), ALLAN));
          c.offset -= dtemp;

          /*
           * Функция корректировки времени ядра, обычно реализуемая
           * системным вызовом Unix adjtime().
           */
          adjust_time(c.freq + dtemp);

          /*
           * Таймер партнёра, вызывающий функцию poll() по завершении.
           */
          while (/* all associations */ 0) {
                  struct p *p;/* Фиктивный указатель на структуру партнёра */

                  if (c.t >= p->nextdate)
                          poll(p);
          }

          /*
           * Один раз в час записывает частоту часов в файл.
           */
          /*
           * if (c.t % 3600 == 3599)
           *   запись c.freq в файл
           */
 }
```

A.5.7. Процесс опроса

```
   /*
    * Параметры и константы процесса опроса.
    */
```

```
   #define UNREACH         12      /* Порог счётчика недоступности */
   #define BCOUNT          8       /* Число пакетов в блоке (burst) */
   #define BTIME           2       /* Интервал между блоками (s) */
```

A.5.7.1. poll()

```
/*
 * poll() - определяет момент передачи пакета для ассоциации p
 */
void
poll(
        struct p *p     /* Указатель на структуру партнёра */
        )
{
        int     hpoll;
        int     oreach;

        /*
         * Эта функция вызывается, когда текущее время c.t догоняет время
         * следующего опроса p->nextdate. Значение p->outdate указывает
         * время последнего вызова функции. Время следующего выполнения
         * p->nextdate задаёт функция poll_update().
         *
         * В широковещательном режиме просто вызывается функция, но
         * только при наличии синхронизации.
         */
        hpoll = p->hpoll;
        if (p->hmode == M_BCST) {
                p->outdate = c.t;
                if (s.p != NULL)
                        peer_xmit(p);
                poll_update(p, hpoll);
                return;
        }

        /*
         * Для manycast начинается с ttl = 1 и значение ttl увеличивается
         * на 1 для каждого опроса, пока не будет найдено MAXCLOCK серверов
         * или ttl не достигнет TTLMAX. При достижении MAXCLOCK опрос
         * прекращается, пока число серверов не упадёт ниже MINCLOCK, после
         * чего процесс повторяется.
         */
        if (p->hmode == M_CLNT && p->flags & P_MANY) {
                p->outdate = c.t;
                if (p->unreach > BEACON) {
                        p->unreach = 0;
                        p->ttl = 1;
                        peer_xmit(p);
                } else if (s.n < MINCLOCK) {
                        if (p->ttl < TTLMAX)
                                p->ttl++;
                        peer_xmit(p);
                }
                p->unreach++;
                poll_update(p, hpoll);
                return;
        }
        if (p->burst == 0) {

                /*
                 * Режим не burst. Регистр доступности сдвигается влево.
                 * Надеемся, что до следующего опроса поступит пакет и
                 * заполнится правый бит регистра.
                 */
                oreach = p->reach;
                p->outdate = c.t;
                p->reach = p->reach << 1;
                if (!(p->reach & 0x7))
                        clock_filter(p, 0, 0, MAXDISP);
                if (!p->reach) {

                        /*
                         * Сервер недоступен, поэтому увеличивается счётчик
                         * unreach. По достижении порога недоступности
                         * интервал опроса удваивается для минимизации
                         * сетевого трафика. Блок пакетов (burst)
                         * передаётся, если это разрешено и
                         * достигнут порог недоступности.
                         */
                        if (p->flags & P_IBURST && p->unreach == 0) {
                                p->burst = BCOUNT;
                        } else if (p->unreach < UNREACH)
                                p->unreach++;
                        else
                                hpoll++;
                        p->unreach++;
                } else {
```

```
                        /*
                         * Сервер доступен. Для интервала опроса задаётся
                         * значение системного интервала опроса. Блок
                         * (burst) передаётся , если это разрешено и
                         * партнёр подходит.
                         */
                        p->unreach = 0;
                        hpoll = s.poll;
                        if (p->flags & P_BURST && fit(p))
                                p->burst = BCOUNT;
                }
        } else {

                /*
                 * В случае значение снижается. При получении отклика
                 * функция clock_filter() вызывает clock_select() для
                 * обработки результатов передачи блока пакетов.
                 */
                p->burst--;
        }
        /*
         * Не передавать в режиме широковещательного клиента.
         */
        if (p->hmode != M_BCLN)
                peer_xmit(p);
        poll_update(p, hpoll);
}
```

A.5.7.2. poll_update()

```
/*
 * poll_update() - обновление интервала опроса для ассоциации p.
 *
 * Эту функцию вызывают packet() и poll(). Поскольку packet() вызывается
 * при получении пакета из сети, а poll() - по тайм-ауту, может возникать
 * «конфликт», который способен сделать некорректным следующий интервал
 * опроса. Такие ситуации считаются пренебрежимо редкими.
 */
void
poll_update(
        struct p *p,            /* Указатель на структуру партнёра */
        int     poll            /* Интервал опроса (log2 s) */
        )
{
        /*
         * Эту функцию вызывают poll() и packet() для определения времени
         * следующего опроса. В блочном (burst) режиме интервал опроса
         * составляет 2 секунды, в ином случае определяется меньшим из
         * значений интервалов опроса хоста и партнёра, но не более MAXPOLL
         * и не менее MINPOLL. Это позволяет заменить долгий интервал более
         * коротким, если требуется быстрый отклик.
         */
        p->hpoll = max(min(MAXPOLL, poll), MINPOLL);
        if (p->burst > 0) {
                if (p->nextdate != c.t)
                        return;
                else
                        p->nextdate += BTIME;
        } else {

                /*
                 * Хотя здесь это не показано, эталонная реализация вносит
                 * интервал опроса небольшой случайный фактор.
                 */
                p->nextdate = p->outdate + (1 << max(min(p->ppoll,
                    p->hpoll), MINPOLL));
        }

        /*
         * Может случиться так, что заданное время уже прошло. В этом
         * случае задаётся время на 1 секунду в будущее.
         */
        if (p->nextdate <= c.t)
                p->nextdate = c.t + 1;
}
```

A.5.7.3. peer_xmit()

```
/*
 * transmit() - передача пакета для ассоциации p
 */
void
peer_xmit(
        struct p *p             /* Указатель на структуру партнёра */
        )
{
        struct x x;             /* Передаваемый пакет */
```

```
        /*
         * Инициализация заголовка и метки времени передачи.
         */
        x.srcaddr = p->dstaddr;
        x.dstaddr = p->srcaddr;
        x.leap = s.leap;
        x.version = p->version;
        x.mode = p->hmode;
        if (s.stratum == MAXSTRAT)
                x.stratum = 0;
        else
                x.stratum = s.stratum;
        x.poll = p->hpoll;
        x.precision = s.precision;
        x.rootdelay = D2FP(s.rootdelay);
        x.rootdisp = D2FP(s.rootdisp);
        x.refid = s.refid;
        x.reftime = s.reftime;
        x.org = p->org;
        x.rec = p->rec;
        x.xmt = get_time();
        p->xmt = x.xmt;

        /*
         * Если идентификатор ключа отличен от 0, передаётся MAC с
         * использованием key ID ассоциации и ключа из локального кэша.
         * Если что-то пошло не так, например, нет доверенного ключа,
         * пакет не передаётся, а ассоциация сбрасывается и прекращает
         * работу до устранения проблем.
         */
        if (p->keyid)
                if (/* p->keyid invalid */ 0) {
                        clear(p, X_NKEY);
                        return;
                }
                x.dgst = md5(p->keyid);
        xmit_packet(&x);
}
```

Адреса авторов

Dr. David L. Mills
University of Delaware
Newark, DE 19716
US
Phone: +1 302 831 8247
EMail: mills@udel.edu

Jim Martin (editor)
Internet Systems Consortium
950 Charter Street
Redwood City, CA 94063
US
Phone: +1 650 423 1378
EMail: jrmii@isc.org

Jack Burbank
The Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Road
Laurel, MD 20723-6099
US
Phone: +1 443 778 7127
EMail: jack.burbank@jhuapl.edu

William Kasch
The Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Road
Laurel, MD 20723-6099
US
Phone: +1 443 778 7463
EMail: william.kasch@jhuapl.edu

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru