IPv4r2

расширение протокола IPv4
до версии (ревизии) два,
описание протокола от 12 декабря 2015 года
полный текст в редакции от 2 марта 2016 года.

Полянин М.А.
12 декабря 2015,
02 марта 2016.

Предисловие.

Протокол IPv4r2 с первого взгляда на его описание выглядит достаточно сложным, но это все потому, что в протоколе IPv4r2 заложен способ его расширения и адаптации к различным конкретным условиям, перечисление этих общих правил создает большой объем текста.

А по своей сущности IPv4r2 это только одна новая опция IPv4 заголовка 88h,08h и такая опция это все что нужно для расширения IPv4 адресации.

Можно запомнить, что IPv4r2 это:

- Легкость преобразования IPv4 системы в IPv4r2
  - улучшение любой IPv4 системы (для которой доступны открытые коды и описание дизайна) до IPv4r2, так что IPv4r2 пакеты смогут приниматься и отправляться, можно выполнить в принципе за один день работы;
  - простота реализации IPv4r2 на реальных машинах обеспечивается "наборами IPv4r2.x" - это подмножества свойств IPv4r2 для конкретных применений, так что частично реализованный протокол остается совместимым при обмене по сети;
- Совместимость с IPv4
  - имеющаяся сетевая инфраструктура IPv4 используется в сетях IPv4r2 без существенных модификаций;
  - "набор IPv4r2.0" позволяет многим IPv4 приложениям работать с IPv4r2 адресатами словно с IPv4, при этом снимая для этих IPv4 приложений ограничения IPv4 адресации;
- Простота получения глобальных IPv4r2 адресов
  - любой провайдер имеет хоть один глобальный IPv4 адрес (через шлюз с таким IPv4 адресом клиенты этого провайдера выходят в интернет) и на основе каждого такого IPv4 адреса провайдер может самостоятельно построить свою отдельную IPv4r2 сеть с сотнями миллиардов глобальных IPv4r2 адресов, этого хватит для каждого клиента.

Описание протокола IPv4r2 разделено на несколько частей:

- первая часть это формальный справочник новых по отношению к IPv4 опций IPv4r2 и описание состава "наборов IPv4r2.x";
- вторая часть содержит причины принятия тех и иных решений в IPv4r2 протоколе и подробное рассмотрение сложных функций IPv4r2 протокола;
- третья часть это примеры реализации IPv4r2 протокола на реальных системах.

Базовое расширение IPv4 до IPv4r2.

Базовое расширение IPv4 до IPv4r2 заключается в добавлении к IPv4 новых опций заголовка IPv4 для поддержки IPv4r2 расширенной адресации и IPv4r2 индексов локальных сокетов шлюза.

Описание IPv4r2 расширенной адресации и IPv4r2 индексов шлюза приведено далее, но для понимания терминов "множество сетей IPv4r2" и "глобальная адресация IPv4r2", о которых говорится в описании опций IPv4r2, необходимо сразу сказать, что:

- Любая сеть IPv4r2 имеет, в терминах IPv4 адресации, многоуровневую иерархическую структуру адреса, т.е. выглядит как несколько вложенных IPv4 адресов. Формат URL "протокол://a1.b1.c1.d1/a2.b2.c2.d2/..." описывает адресацию в сети IPv4r2. При этом IPv4 адрес a1.b1.c1.d1, который указан в IPv4 заголовке это, в терминах IPv4r2 адресации, адрес в корневой сети IPv4 (уровень корневой сети IPv4).
- Каждый адрес в корневой сети IPv4 определяет точку входа в IPv4r2 сеть, таким образом, сетей IPv4r2 получается множество и формат IPv4 адресов внутренних уровней в этих сетях зависит от структуры конкретной IPv4r2 сети. Если корневая сеть IPv4 для адреса a1.b1.c1.d1 является глобальной, то и вся эта сеть IPv4r2 может быть адресована глобально, на уровне интернет (это и есть та глобальная адресация, про которую говорится в описании опций IPv4r2).

Расширенная IPv4r2 адресация подразделяется на несколько типов:

- основная IPv4r2 адресация:
  - базовая;
  - обобщенная;
- дополнительная IPv4r2 адресация сетей пользователя;
- дополнительная IPv6 адресация в сетях IPv4r2.

Основная базовая IPv4r2 адресация.

Для добавления свойств IPv4r2 к заголовку IPv4 используется опция 0x88 (136) заголовка IPv4 специального формата. В терминах IPv4r2 опция 0x88 называется op1. По правилам опций заголовка IPv4, для любой опций IPv4 после октета (в этом

тексте октет эквивалентен байту) тип (для op1=0x88) идет октет длина, хранящий размер опции, размер опций включает в себя и первые два байта опции (включает байты тип 0x88 и длина).

Базовые варианты кодирования новых IPv4r2 опций:

- <op1>[8]<длина=8>[8]<флаги>[3]<адрес>[45]
  IPv4r2 асимметричная адресация 45 бит
- <op1>[8]<длина=12>[8]<флаги>[3]<адрес>[45]<индекс>[32]
  IPv4r2 асимметричная адресация 45 бит
  IPv4r2 индекс локального сокета шлюза 32 бита
- <op1>[8]<длина=16>[8]<адрес источника>[48]<адрес назначения>[48]<индекс>[16]
  IPv4r2 адресация 48 бит
  IPv4r2 индекс локального сокета шлюза 16 бит

Для IPv4r2 асимметричной адресации 45 бит, трех- битовое поле флагов обозначает:

- 010b
  расширение адреса источника
- 100b
  расширение адреса назначения

Для IPv4r2 адресации поле <индекс>[число бит] имеет следующий формат:

- <флаг>[1]<значение индекса>[остальное]
  бит флаг обозначает принадлежность шлюза адресу источника или назначения:
  - 0b
    индекс для адреса источника
  - 1b
    индекс для адреса назначения

Описание базовых вариантов кодирования IPv4r2 опций.

"IPv4r2 асимметричная адресация 45 бит" (т.е. расширение адреса на 45 бит, общий размер адреса 77 бит) это основной вид глобальной адресации пользовательских серверов в сетях IPv4r2 специального типа, когда IPv4r2 источник подключен к сети IPv4r2 через IPv4 шлюз провайдера (источник использует расширение адреса назначения).

"IPv4r2 асимметричная адресация 45 бит с индексом шлюза" автоматически используется при доставке пакетов в формате "IPv4r2 асимметричная адресация 45 бит" на участке сети между шлюзом провайдера и пользовательским сервером, в том случае, когда IPv4 адрес шлюза провайдера перегружен локальными подключениями от клиентов провайдера. Практически размер

IPv4r2 заголовка от этого возрастает на 4 байта.

"IPv4r2 адресация 48 бит" (т.е. расширение адреса на 48 бит, общий размер адреса 80 бит) это расширенный вид 45 битной основной глобальной адресации в сетях IPv4r2 специального типа, когда оба IPv4r2 адреса уникальны на глобальном уровне, поэтому IPv4 шлюз провайдера не обязателен, но может быть использован. Для преобразования адреса 45 бит в адрес 48 бит и обратно используются специальные правила, которые рассмотрены далее в разделе "полное описание IPv4r2".

Рекомендуемое значение глобальной уникальной адресации для сетей IPv4r2 - 77 бит.

Общий формат кодирования IPv4r2 опций.

Тип расширения IPv4r2, который задается опцией IPv4r2 op1=136, определяется размером опции, а также дополнительными флагами в области данных опции, флаги применяются тогда, когда одному размеру опции соответствует несколько разных расширений.

Список дополнительных кодировок для IPv4r2 опции op1:

- <op1>[8]<длина=4>[8]<номер потока>[16]
  IPv4 зарезервировано для номера потока
- <op1>[8]<длина=6>[8]<индекс>[32]
  IPv4r2 индекс локального сокета шлюза 32 бит
- <op1>[8]<длина=10>[8]<флаги>[3]<адрес>[45]<индекс>[16]
  IPv4r2 асимметричная адресация 45 бит
  IPv4r2 индекс локального сокета шлюза 16 бит
- <op1>[8]<длина=14>[8]<адрес источника>[48]<адрес назначения>[48]
  IPv4r2 адресация 48 бит
- <op1>[8]<длина=18>[8]<адрес источника>[48]<адрес назначения>[48]<индекс>[32]
  IPv4r2 адресация 48 бит
  IPv4r2 индекс локального сокета шлюза 32 бит

Для IPv4r2 асимметричной адресации 45 бит, трех- битовое поле флагов дополнительно обозначает:

- 001b
  индекс локального сокета шлюза
  эти флаги используются для опций асимметричной адресации, например
  - <op1>[8]<длина=8>[8]<флаги>[3]<адрес>[45]
    при таких флагах эта опция используется для задания индекса шлюза

Для дополнительных IPv4r2 расширений, когда одному размеру опции op1 соответствует несколько разных расширений, но флаги

задать нельзя, используется опция IPv4r2 op2 (числовое значение еще не зафиксировано, предположительно 0x8A).

Список базовых кодировок для IPv4r2 опции op2:

- <op2>[8]<длина=4>[8]<индекс>[16]
  IPv4r2 индекс локального сокета шлюза 16 бит
- <op2>[8]<длина=12>[8]<адрес источника>[40]<адрес назначения>[40]
  IPv4r2 короткая адресация 40 бит
- <op2>[8]<длина=16>[8]<адрес источника>[40]<адрес назначения>[40]<индекс>[32]
  IPv4r2 короткая адресация 40 бит
  IPv4r2 индекс локального сокета шлюза 32 бит

IPv4r2 адресация 40 бит (т.е. расширение адреса на 40 бит, общий размер адреса 72 бит) это короткий вид 45 битной основной адресации в сетях IPv4r2 специального типа, когда оба IPv4r2 адреса уникальны на глобальном уровне. Практически это позволяет уменьшить размер IPv4r2 заголовка на 4 байта по сравнению с такой же адресацией 48 бит, когда IPv4 шлюз провайдера не используется и если структура этой сети IPv4r2 допускает 72 битную глобальную уникальную адресацию, эта возможность зависит от структуры конкретной сети IPv4r2. Для преобразования адреса 45 бит в адрес 40 бит и обратно используются специальные правила, которые рассмотрены далее в разделе "полное описание IPv4r2".

Список дополнительных кодировок для IPv4r2 опции op2:

- <op2>[8]<длина=14>[8]<адрес источника>[40]<адрес назначения>[40]<индекс>[16]
  IPv4r2 короткая адресация 40 бит
  IPv4r2 индекс локального сокета шлюза 16 бит

Основная обобщенная IPv4r2 адресация.

Обобщенная IPv4r2 адресация это (по сравнению с базовой IPv4r2 адресацией) альтернативный и менее эффективный способ указания адресов в сетях IPv4r2.

Для дополнительных IPv4r2 расширений, когда одному размеру опций op1, op2 соответствует несколько разных расширений, но флаги задать нельзя, используется опция IPv4r2 op3 (числовое значение еще не зафиксировано, предположительно 0x8B).

Список основных адресных кодировок для обобщенной IPv4r2 адресации:

- <op1>[8]<длина=3+>[8]<формат опции=1>[1]<флаг адреса>[1]<младший байт адреса>[6]<адрес>[0+]
  длина больше равна 3 и всегда нечетная
  IPv4r2 асимметричная обобщенная адресация 6+ бит
- <op2>[8]<длина=7+>[8]<формат опции=1>[1]<флаг адреса>[1]<младший байт адреса>[6]<адрес>[0+]<индекс>[32]
  длина больше равна 7 и всегда нечетная

IPv4r2 асимметричная обобщенная адресация 6+ бит
IPv4r2 индекс локального сокета шлюза 32 бит

- <op3>[8]<длина=4+>[8]<формат опции=1>[1]<размер индекса>[2]<размер адреса источника>[5]<адрес источника>[0+]<адрес назначения>[0+]<индекс>[0+]
  длина больше равна 4
  IPv4r2 обобщенная адресация
  размер адреса источника в байтах 0..31
  размер индекса локального сокета шлюза в байтах 0..3
  размер адреса назначения = длина-3-"размер адреса источника"-"размер индекса"

Формат флага адреса для асимметричной обобщенной адресации:

- флаг адреса
  - 0b - расширение адреса источника
    1b - расширение адреса назначения

IPv4r2 асимметричная адресация 6+ бит это обобщенная глобальная IPv4r2 адресация при работе через IPv4 шлюз провайдера. Есть варианты кодирования с индексом локального сокета и без такого индекса.

Распределение байтов поля адреса обобщенной IPv4r2 адресации по уровням IPv4r2 адресации имеет сложную структуру и отличается от распределения битов поля адреса в основных режимах IPv4r2 адресации (в режимах IPv4r2 адресации фиксированного 40, 45 и 48 бит размера). Обобщенная IPv4r2 адресация допустима, но практического смысла не имеет. Подробнее смотри в разделе "полное описание IPv4r2".

Список дополнительных адресных кодировок для обобщенной IPv4r2 адресации:

- <op2>[8]<длина=5+>[8]<формат опции=0>[1]<флаг адреса>[1]<младший байт адреса>[6]<адрес>[0+]<индекс>[16]
  длина больше равна 5 и всегда нечетная
  IPv4r2 асимметричная обобщенная адресация 6+ бит
  IPv4r2 индекс локального сокета шлюза 16 бит

Дополнительная IPv4r2 адресация сети пользователя.

Дополнительная IPv4r2 адресация пользовательской сети позволяет на один централизованно выделенный пользователю IPv4r2 адрес оперативно размещать дополнительные ресурсы пользователя, глобально адресуемые и уникальные путем привязки к этому выделенному IPv4r2 адресу.

Дополнительная адресация пользовательской сети это отдельное адресное пространство, которое не может быть адресовано через основную (базовую или обобщенную) адресацию, именно поэтому в нем можно выделять ресурсы самому владельцу IPv4r2

адреса.

Список кодировок IPv4r2 адресации пользовательской сети:

- <op3>[8]<длина=3+>[8]<формат опции=001>[3]<флаг адреса>[1]<старший байт адреса>[4]<адрес>[0+]
  длина больше равна 3
  IPv4r2 асимметричная адресация расширения пользовательской сети
  формат байтов поля адреса: от старшего байта к младшему
- <op3>[8]<длина=4+>[8]<формат опции=0001>[4]<число байт адреса источника>[4]<байты источника>[0+]<байты назначения>[0+]
  длина больше равна 3+1=4 и меньше равна 3+4+4=11
  IPv4r2 адресация расширения пользовательской сети
  число байт назначения = длина-3-"число байт источника"
  формат байтов поля адреса: от старшего байта к младшему

Формат флага адреса для IPv4r2 адресации пользовательской сети:

- флаг адреса
  - 0b - расширение адреса источника
    1b - расширение адреса назначения

Дополнительная IPv6 адресация в сетях IPv4r2.

Для самого протокола IPv4r2 адресация IPv6 не нужна, но приходится учитывать тот практический фактор, что сеть IPv6 уже реально существует и полезно иметь доступ к ресурсам этой сети IPv6 из сети IPv4r2, при этом такой IPv6 компьютер получит IPv4 (с опциями IPv4r2), а не IPv6 пакет.

IPv6 адресация это новое адресное пространство IPv4r2, отдельное от основного и пользовательского адресных пространств IPv4r2, по типу IPv4r2 адресации аналогичное принципам дополнительной IPv4r2 адресации сети пользователя.

Список кодировок IPv6 адресации хостов в сетях IPv4r2 128 бит:

- <op1>[8]<длина=20>[8]<флаг адреса>[1]<резерв 0>[15]<адрес IPv6>[128]
  IPv4r2 асимметричная IPv6 адресация хостов в сетях IPv4r2 128 бит
  формат байтов поля адреса: от старшего байта к младшему

Формат флага адреса для асимметричной IPv6 адресации хостов в сетях IPv4r2:

- флаг адреса
  - 0b - расширение адреса источника

1b - расширение адреса назначения

Поскольку правила записи URL для IPv6 адреса определены в стандарте IPv6 и:

- изменения в стандарте IPv6 порождали бы необходимость модификаций стандарта IPv4r2;
- был бы не решен вопрос с совместимостью форматов URL IPv6 и IPv4r2;

то URL IPv6 адреса для IPv4r2 записывается по собственным IPv4r2 правилам, имеющим вид "четыре фиксированных адреса IPv4 (a/b/c/d), старшая часть адреса a/ первая".

Также IPv4 система, которую модифицируют до IPv4r2, не имеет функции разбора IPv6 адреса по правилам IPv6 и формат "четыре адреса IPv4" позволяет не создавать такой сервис разбора URL IPv6 для протокола IPv4r2.

Подсети IPv4r2.

В IPv4r2 сетях, подсети вида "IPv4r2 адрес нулевого хоста подсети"/:"число бит маски подсети", могут формироваться без выделения IPv4r2 адресов на широковещательный адрес и маршрут к сети, для этого вместо выделенного IPv4r2 адреса применяются IPv4r2 опции, которые кодируют:

- число бит маски (или обратной маски) такой подсети для IPv4r2 адресата;
- флаг широковещательный запрос для IPv4r2 адреса назначения.

Список кодировок для IPv4r2 подсетей:

- <op3>[8]<длина=4..5>[8]<формат опции=01>[2]<флаги маски>[6]<маска источника>[0+]<маска назначения>[0+]
  длина от 4 до 5
  размер поля маска источника и маска назначения один байт, поле содержит число бит маски подсети вида /:"число бит маски подсети"
  - флаги маски сети
    000001b - есть поле число бит маски сети источника, хранящее маску
    000010b - есть поле число бит маски сети источника, хранящее обратную маску
    000100b - есть поле число бит маски сети назначения, хранящее маску
    001000b - есть поле число бит маски сети назначения, хранящее обратную маску
    010000b - это IPv4r2 ответ от службы сети источника
    100000b - это широковещательный IPv4r2 запрос к сети назначения

Если установлен флаг "широковещательный IPv4r2 запрос к сети назначения", то IPv4r2 адрес в поле назначения хранит не адрес хоста, а после применения маски, этот адрес указывает на сеть назначения.

Если установлен флаг "IPv4r2 ответ от службы сети источника", то IPv4r2 адрес в поле источника хранит не адрес хоста, а

после применения маски, этот адрес указывает на сеть источника.

Поле маски включает в себя и пространство дополнительной IPv4r2 адресации пользовательской сети - основное и пользовательское адресные пространство для маски подсети объединяется, основное адресное пространство для маски подсети идет первым.

Команды для IPv4r2 шлюза провайдера.

Для того чтобы правильно транслировать работу IPv4r2 через глобальный IPv4 шлюз применяются эти команды для IPv4r2 шлюза провайдера.

Только тот шлюз, который имеет глобальный IPv4, обрабатывает эти команды и только от своей локальной сети и не пропускает эти команды в глобальную сеть (забивает кодом 1 или удаляет из списка опций). Эти команды генерирует клиент, который знает какой протокол IPv4r2.x применяется при обмене в каждом конкретном случае.

Правила использования индексов шлюзов при трансляции через шлюз провайдера:

- по умолчанию для IPv4r2 пакета:
  - задан IPv4r2 адрес из локальной сети (IPv4r2 источник исходящий)
    IPv4 трансляция не проводится
  - задан IPv4 адрес в глобальной сети (IPv4 назначение исходящий)
    IPv4 трансляция проводится по правилам IPv4 (транслировать UDP/TCP), без индексов шлюза
  - задан только IPv4r2 адрес в глобальной сети (IPv4r2 назначение исходящий)
    IPv4 трансляция проводится по правилам IPv4 (транслировать UDP/TCP), индексы на усмотрение шлюза *(т.е. даже с индексом шлюза сетевой мост IPv4r2 не будет создан, а индекс позволяет шлюзу не истощать IPv4 порты для исходящих запросов)*

По умолчанию глобальным шлюзам IPv4r2 разрешено использовать индексы при трансляции для IPv4r2 адресатов, но не разрешено создавать сетевой мост, т.к. если нужны входящие подключения, то локальный компьютер может просто указать свой IPv4r2 адрес.

Эти умолчания могут быть переопределены или подтверждены командой для IPv4r2 шлюза провайдера.

Список кодировок IPv4r2 команд для шлюза провайдера:

- <op3>[8]<длина=3>[8]<формат опции=0001>[4]<код команды>[4]
  длина равна 3
  IPv4r2 команда шлюзу провайдера от адресата локальной сети провайдера

Список кодов команд:

- 0x00
  для данного IPv4r2 пакета, адресат из локальной сети IPv4
  использовать IPv4 трансляцию от источника в локальной сети для IPv4r2 адреса назначения
  IPv4r2.0 совместимость, позволяет IPv4 софту работать
  запрещаются индексы шлюза, т.к. сервер не может их обработать
- 0x01
  для данного IPv4r2 пакета, адресат из локальной сети IPv4 или оба адресата IPv4
  использовать IPv4r2 трансляцию от источника в локальной сети для IPv4r2 адреса назначения
  IPv4r2.1 совместимость, позволяет шлюзу не истощать IPv4 порты для исходящих запросов
  разрешаются индексы шлюза и обязательно с NAT трансляцией по правилам IPv4
  запрещается создание IPv4r2 моста с авто IPv4r2 адресом,
  недопустимы IPv4r2 входящие по такому мосту,
  допустимы только запросы на открытый порт UDP/TCP

остальные коды зарезервированы.

Во всех списках эффективных порядков адресных опций (адресных списках) IPv4r2.x, команда для шлюза замыкает список расширения адреса источника (или заменяет эти расширения при их отсутствии) для исходящих запросов от локальной сети в глобальную.

Вспомогательные IPv4r2 опции.

Список вспомогательных IPv4r2 опций:

- <op2>[8]<длина=6>[8]<отметка времени>[32]
  Для установления времени жизни IPv4r2 пакета, источник IPv4r2 может отмечать свои пакеты отметкой времени синхронизированного со своим корневым a1.b1.c1.d1 (UTC время в секундах). Для этого IPv4r2 определяет механизм опроса источника времени в корневой IPv4 сети:
  - корневой узел a1.b1.c1.d1 сети IPv4r2 обязан реализовать ответы о времени на запрос от любого другого корневого IPv4 не реже раза в минуту после первого ответа для каждого такого корневого адреса;
  - IPv4r2 получатель обращается за временем a1.b1.c1.d1 источника к своему корневому a1.b1.c1.d1.

Комбинации новых IPv4r2 опций.

Опции op1, op2, op3 и флаги могут комбинироваться по их смыслу, при этом нельзя:

- аддитивно наращивать расширение одного и того же адреса, несколько раз применяя опции такого расширения;
- аддитивно наращивать индекс одного и того же шлюза с помощью полей плюс флагов в одной опции или с помощью нескольких опций;

- комбинировать обобщенную и базовую адресацию IPv4r2 для одного и того же адреса;

Например, можно комбинировать <op1>[8]<длина=14>[8] и <op1>[8]<длина=6>[8] для 32 индексации локальных сокетов шлюза и 48 битного расширения адреса, тем увеличивая IPv4r2 заголовок до 20 байт, заместо более компактного 16 байтового формата <op1>[8]<длина=16>[8] с 16 битной индексацией локальных сокетов шлюза, если 16 бит индексов сокета шлюза не хватает.

С точки зрения эффективности маршрутизации, лучше задавать расширение адресов и индексы шлюза только одной опцией, идущей в заголовке первой. Опция дополнительной IPv4r2 адресации пользовательской сети и дополнительной IPv6 адресации (рекомендуется задавать второй) и опция IPv4r2 подсетей (рекомендуется задавать третьей) обычно анализируются только конечными IPv4r2 адресатами, промежуточные IPv4r2 маршрутизаторы их игнорируют.

IPv4r2 пакеты с ошибочными адресными опциями не могут быть доставлены и будут отброшены IPv4r2 маршрутизаторами и адресатами.

Протоколы "IPv4r2.x".

Протокол IPv4r2 в целом не сложный, но его полная реализация довольно объемная, при том что на практике 90% возможностей протокола IPv4r2 в каждом клиенте не будет использовано никогда, поэтому возникает противоречие между целостностью IPv4r2 протокола и целям создания IPv4r2 протокола для упрощения расширения адресации IPv4.

С другой стороны, задание опциональных возможностей протокола приведет к тому, что IPv4r2 адресаты будут постоянно использовать несовместимые опции и не смогут установить между собой связь.

Чтобы решить эту проблему выделяются части IPv4r2 протокола:

- базовая часть IPv4r2.0;
- первый набор IPv4r2.1;
- второй набор IPv4r2.2;
- и т.д.;

так что:

- обменивающиеся по сети машины должны соответствовать одинаковой части реализуемых возможностей IPv4r2;
- IPv4r2.1 соответствует битовой маске 1, IPv4r2.2 соответствует битовой маске 2 и т.д.;
- протокол объединяющий .1 и .2 соответствует битовой маске 3 и значит имеет номер 3;
- все IPv4r2.x входят в "полный IPv4r2";
- протокол IPv4r2.0 должен быть реализован всеми IPv4r2 устройствами.

Протокол "IPv4r2.0" (базовый).

Протокол IPv4r2.0 предназначен для работы IPv4 приложений поверх IPv4r2 расширения адресации.

Для того чтобы IPv4 приложения могли использовать глобальные IPv4r2 адреса, IPv4r2 адреса отображаются на локальные IPv4 адреса.

С помощью IPv4r2.0 обычные пользователи получают возможность легко размещать свои серверные ресурсы в сети интернет и легко получать доступ к другим IPv4r2.0 ресурсам с помощью IPv4 приложений, обходя ограничения IPv4 адресации. Для такого типового применения и разработан "IPv4r2.0".

Поскольку IPv6 адресация уже применяется на практике, протокол IPv4r2.0 позволяет IPv4 приложениям адресовать компьютеры подключенные к IPv4r2 сетям, для которых известен их глобальный IPv6 адрес, при этом такой IPv6 компьютер получит IPv4, а не IPv6 пакет.

Типовое использование IPv4r2.0 следующее:

- работа в сети интернет
  IPv4r2 сервер имеет глобальный IPv4r2 адрес;
  IPv4r2 клиент не имеет глобального IPv4r2 адреса и обращается к этому IPv4r2 серверу через шлюз провайдера с IPv4 глобальным адресом шлюза;
- работа в специальных и локальных IPv4r2 сетях
  IPv4r2 сервер и IPv4r2 клиент имеют глобальные или локальные IPv4r2 адреса;

Возможности протокола IPv4r2.0 (все остальное, что не входит в IPv4r2.0 входит в полную реализацию IPv4r2):

- 45 бит асимметричный базовый адрес опцией <op1=0x88>[8]<8>[8]<флаги>[3]<адрес>[45] (B45)
- 12 бит пользовательское адресное пространство <op3=0x8B>[8]<4>[8]<флаги>[4]<адрес>[12] (U12)
- 128 бит IPv6 адресация в сетях IPv4r2 <op1=0x88>[8]<20>[8]<флаги>[16]<адрес>[128] (V6)
- команды для IPv4r2 шлюза <op3=0x8B>[8]<3>[8]<флаги>[4]<команда>[4] (GC)

Системные IPv4r2.0 сервисы:

- совместимость с IPv4 сетевым оборудованием
  - UDP инкапсуляция исходящих IPv4r2 пакетов от IPv4 адреса исходный:4 на удаленный IPv4 адрес 192.88.99.1:4 и обратная распаковка входящих UDP по сигнатуре 55AA4200
- совместимость с IPv4 программами
  - NAT отображение IPv4r2 адресов на IPv4 локальные адреса по локальной таблице IPv4r2 ⇔ IPv4
  - статическая часть NAT таблицы

    - DNS динамическое отображение IPv4r2 адресов на IPv4 в динамическую часть NAT таблицы
    - генерация команд для IPv4r2 шлюза провайдера "использовать IPv4 правила трансляции"

Протокол IPv4r2.0 позволяет:

- в 45 бит обращаться к базовым адресам IPv4r2;
- в 12 бит обращаться к пользовательским адресам IPv4r2;
- в 128 бит обращаться к IPv6 адресатам в сети IPv4r2;
- многим IPv4 приложениям работать без изменений в сетях IPv4r2.

Для протокола IPv4r2.0 эффективный порядок адресных опций (адресный список), который превращает работу с IPv4r2.0 пакетом к подобию работы с IPv6 пакетом, такой:

- источник, указатель начиная со смещения IPv4 заголовка 20:
    - B45, если найдено запись + 8 к указателю, иначе следующее сравнение
    - U12, если найдено запись + 4 к указателю, иначе следующее сравнение
    - V6, если найдено запись + 20 к указателю, иначе следующее сравнение
    - (команда для IPv4r2 шлюза GC)
- назначение, указатель начиная с текущего смещения:
    - B45, если найдено запись + 8 к указателю, иначе следующее сравнение
    - U12, если найдено запись + 4 к указателю, иначе следующее сравнение
    - V6, если найдено запись + 20 к указателю, иначе следующее сравнение

если на любой сравниваемой опции просмотрели весь адресный список и получили иную опцию или опции кончились, значит адреса IPv4r2.0 получены.

Для того чтобы это сработало, пакет должен прибыть по сети IPv4r2.0, это можно быстро определить по IPv4 адресу сети источника "(адрес XOR маска сети) AND маска сети" - входит ли этот IPv4 адрес в список "сетей IPv4r2.0".

(см. дополнительное описание IPv4r2.0 далее)

Протокол "IPv4r2.1".

По сути это те части базового протокола IPv4r2.0, с которыми могут работать только IPv4r2 приложения, невозможно транслировать эти IPv4r2 опции на IPv4 адреса для IPv4 приложений.

Возможности протокола IPv4r2.1 (все остальное, что не входит в IPv4r2.1 входит в полную реализацию IPv4r2):

- для IPv4r2 клиента и сервера:

    - отметка времени опцией <op2=0x8A>[8]<6>[8]<отметка времени>[32] (T32)
- для IPv4r2 сервера дополнительно:
    - 16 бит индекс шлюза опцией <op2=0x8A>[8]<4>[8]<индекс>[16] (I16)
    - 32 бит индекс шлюза опцией <op1=0x88>[8]<6>[8]<индекс>[32] (I32)

Для протокола IPv4r2.1 эффективный порядок адресных опций (адресный список) отличается от IPv4r2.0 тем, что после сравнения на V6 идет сравнение на I16 и I32, так:

- I16, если найдено запись + 4 к указателю, иначе следующее сравнение
- I32, если найдено запись + 6 к указателю, иначе следующее сравнение

Протокол "IPv4r2.2".

Возможности протокола IPv4r2.2 (все остальное, что не входит в IPv4r2.2 входит в полную реализацию IPv4r2):

- 70 бит асимметричный обобщенный адрес опцией <op1=0x88>[8]<11>[8]<флаги>[2]<адрес>[70] (G70)
- 102 бит асимметричный обобщенный адрес опцией <op1=0x88>[8]<15>[8]<флаги>[2]<адрес>[102] (G102)
- 38 бит асимметричный обобщенный адрес опцией <op1=0x88>[8]<7>[8]<флаги>[2]<адрес>[38] (G38)

Протокол IPv4r2.2 позволяет:

- в 70 бит обращаться к неиспользуемым в базовой адресации адресам IPv4r2;
- в 102 бит ко всем IPv4r2 адресам формата (128 + 6) бит;
- а также иметь короткое IPv4r2 основное адресное пространство 38 бит.

Для протокола IPv4r2.2 эффективный порядок адресных опций (адресный список) отличается от IPv4r2.0 тем, что после сравнения на V6 идет сравнение на G70, G102 и G38 бит адресные опции.

Протокол "IPv4r2.3".

Для протокола IPv4r2.3 эффективный порядок адресных опций состоит из таких списков для .0, .1, .2; т.е. для объединяющих битовые маски протоколов, список всегда сортируется по добавлениям к IPv4r2.0 в порядке битовой маски от младшего к старшему: (B45, U12, V6), (I16, I32), (G70, G102, G38) адресные опции.

Причины, лежащие в основе выбора свойств протокола IPv4r2.

Введение.

Перейти с IPv4 на IPv6 это все равно что поменять уровень и частоту напряжения в сети, после того как это повсеместно

стало 220 Вольт и 50 Гц, зачем это нужно затевать пользователю? Чтобы у него все отключилось?

В общем случае, если спросить "модернизировать ли старые проблемы или выбрать новый формат", то правильный ответ будет "выбрать новый формат". Но в конкретном частном случае надо смотреть, что же именно модернизируется.

Протокол IPv6, каким бы он ни был хорошим, а он на самом деле не так и хорош, как может показаться, имеет одну проблему - тотальную несовместимость с сетями IPv4, с прежними программами и оборудованием для IPv4. Недопустимо вносить такую несовместимость (на уровне недоставки пакетов и невозможности адресации) без наличия веских причин, при которых продолжать нормальную работу в текущих условиях просто невозможно.

Говоря о тотальной несовместимости, мы имеем ввиду не только физическую возможность или невозможность доставки нового IPv6 пакета по сетям IPv4, но и сложность работ по модификации прежних IPv4 программ с открытым кодом на стороне пользователя, которые надо выполнить при введении нового IPv6 протокола.

Какие же цели достигаются при модификации IPv4 до IPv4r2 вместо полной его замены на IPv6?

Модифицируя IPv4 мы сохраняем полную совместимость с аппаратным и программным оборудованием, действовавшим до этого 20 лет, сохраняем так, что нет нужды его физически переделывать или сложность работ по модификации прежних IPv4 программ с открытым кодом на стороне пользователя минимальна.

Метод модификации IPv4 делает модифицированный протокол IPv4r2 "внутренне несовершенным", но это проблема от того, что исходный IPv4 изначально был внутренне несовершенным, чтобы его могли использовать на глобальном уровне (от осины не родятся апельсины) и новая версия IPv6, предложенная "олигархами", будет на глобальном уровне столь же убога как и IPv4, и эти проблемы IPv6 так же предсказуемы, как были предсказуемы проблемы IPv4.

В отличие от каких-то иных вещей, для такого протокола, как межсетевой протокол (протокол интернета) важно, чтобы он был принят всеми участниками, это вроде как признание золота в качестве ценности. Лучший способ этого добиться для IPv4r2 - реализовать такой IPv4r2 протокол на на тех компьютерных системах, которые используются пользователем так, чтобы пользователь мог его задействовать, если это ему интересно.

Политизированность и рыночность технического вопроса.

Я вовсе не политизирую технический вопрос, но действия разных "консорциумов и комитетов по стандартизации" за последние десятки лет, среди них такие действия как принятие форматов ATA, CD/DVD, USB, PCI, EFI твердо указывают на то, что за этими стандартами стоят вовсе не технические причины, а стандарты принимаются так, чтобы из них можно было бы извлекать прибыль, чтобы введением стандартов создавалась бы определенная проблема, ключи в решении которой были бы у авторов стандарта.

Права разработчиков конечно должны защищаться, но не ущерб же соединению между собой компонентов предназначенных для общей работы, т.е. контроль за соблюдением прав тех, кто вложил деньги в разработку стандарта, должен лежать на уровне производства и допуска лицензионного товара к продаже, а не в деятельности самих приборов. Разного рода соглашения между производителями могут легко нарушаться на уровне незаконной продажи и серых товаров.

В общем вопрос прав на интерфейсы не простой. Те производители, кто выполняет серьезную работу по формированию интерфейсов и рынка устройств с таким интерфейсом не хотят, чтобы другие производители просто пришли и стали делать совместимые копии. А пользователь не хочет, чтобы интерфейсы разных производителей были бы несовместимы между собой. Погоня производителя за защитой его прав приводит к серьезным техническим проблемам в работе устройств. Это в целом проблема некачественного общественного устройства, противоречивого и конфликтного, где извлечение прибыли как угодно и откуда угодно является единственной целью и смыслом деятельности каждого.

В общем те производители, кто не оплатил процесс создания общего интерфейса и формирования рынка потребителей такого интерфейса должны заплатить именно за это при использовании такого стандарта в своих изделиях. Оплата за общий стандарт не может быть использована для извлечения вечной прибыли для создателя стандарта или для конкурентной борьбы.

В целях борьбы с монополизмом производитель должен быть не вправе закрывать стандарты, не вправе лицензировать принципы, но в праве требовать от других участников рынка оплатить все его расходы по созданию этого стандарта, по формированию рынка для этого интерфейса или этого принципа, включая все его риски по провалу, которые у него были при создании такого рынка и он нес их в одиночку. Другими словами, быть вторым производителем, ждущим как пойдет дело у первого, не должно быть выгодно. Также первый производитель может иметь определенное время на торговое преимущество на новом рынке, но не более чем на 50% по объему продаж, чтобы быть первым было выгодно с точки зрения реализации продукции.

В итоге, с такими лицензиями все плохо. При социализме таких проблем нет.

Технические задачи IPv4r2.

IPv4 это такой протокол сетей, который позволяет адресовать компьютеры в сети. В глобальной сети интернет, имея только 32 бита на такой сетевой адрес, IPv4 исчерпал свои возможности по такой сетевой адресации. Также IPv4, инкапсулируя в себя фрагментацию, подсчет контрольных сумм и даже маршрутизацию, нарушает принцип разделения задач по уровням сетевого взаимодействия, подобным уровням сетевой модели OSI.

Задачей протокола IPv4r2, как сетевого протокола, является доставка пакетов, путем:

- адресации конечной и начальной точек сетевого пути;
- инкапсуляции доставляемых данных;

в этом смысле фрагментация и доставка фрагментов, подсчет контрольной суммы и маршрутизация всех пакетов не является задачей IPv4r2.

Для реализации каждой из таких функций для IPv4r2 протокола должен использоваться внешний специализированный протокол-обертка, сложность которого зависит от сложности решения этих задач на данном участке пути при доставке IPv4r2 пакета и в который пакет IPv4r2 должен быть вложен как обычные данные. Так что на простых путях доставки и в малых сетях такие внешние протоколы-обертки могут не понадобиться совсем.

Тем не менее, IPv4r2 позволяет прямо общаться с традиционными IPv4 сетями, допуская подсчет контрольной суммы, фрагментацию и доставку фрагментов, маршрутизацию по правилам IPv4 сети.

Таким образом для обеспечения этих противоречивых требований IPv4 и IPv4r2, протокол IPv4r2 может работать в двух основных режимах:

- в режиме совместимости с IPv4
  в котором обеспечивается только расширенная адресация IPv4r2 при полной совместимости с IPv4;
- в полном, основном режиме
  в котором обеспечивается полная оптимальная функциональность IPv4r2.

Для упрощения на стороне пользователя управления этими свойствами IPv4r2 для разных IP адресов разных типов, адреса традиционного IPv4 динамически могут отображаться на локальные адреса машины так, что при обращении в эту локальную IPv4 подсеть, протокол IPv4r2 работает в режиме совместимости с IPv4 и не задействует свойства IPv4r2, несовместимые с IPv4.

Предполагается, что IPv4r2 будет использоваться:

- как межсетевой протокол глобальной сети с малыми пакетами и большими IPv4r2 адресами;
- как сетевой протокол локальной сети с большими IPv4r2 пакетами и малыми IPv4 адресами;
- возможны обратные комбинации в случае специальных сетей.

Сложность модификации IPv4 устройства для обеспечения работы IPv4r2.

Работая в основном режиме, протокол IPv4r2 не является 100% совместимым с IPv4, поэтому могут возникать задачи модификации старых IPv4 устройств для поддержки ими работы IPv4r2 протокола в основном режиме.

Модификация модификации рознь. Рассматривая изменения в сетевых пользователях IPv4, которые нужно в них внести для обеспечения их совместимости с IPv4r2, мы будем изучать вопрос "какие изменения надо внести в программу обслуживания IPv4 с открытым исходным кодом для обеспечения совместимости с IPv4r2, надо ли при этом:

- менять ядро ОС, глобально и тяжело адаптируя новое ядро под имеющуюся аппаратуру;
- тяжело добавлять новые сетевые службы от нового ядра на старое ядро;
- легко добавлять новые фильтры на старые сетевые службы;

- легко менять константы и способы обработки полей опций в старых сетевых службах;
- выполнять т.п. простые работы;

оценивая все эти работы по сложности в модификации и в последующей отладке".

Например, протокол IPv4r2 имеет резервное значение 0xFFFF в поле контрольной суммы IPv4 заголовка. Для исправления программ, обслуживающих оборудование поддерживающее исходный IPv4, с целью внесения совместимости по интерпретации IPv4r2 поля контрольной суммы, изменения будут откровенно косметическими (по сравнению с ситуацией внедрения IPv6), даже "решение в лоб" путем:

- фильтрации всех входящих IPv4 пакетов с полем контрольной суммы 0xFFFF на предмет подсчета контрольной суммы;
- фильтрации всех исходящих IPv4 пакетов установкой поля контрольной суммы в значение 0xFFFF;

без учета эффективности, равной занесению правильных значений в момент создания IPv4r2 заголовка.

Так вот, в этом примере изменения для программ с открытым исходным кодом, связанные с использованием резервного значения 0xFFFF в поле контрольной суммы, будут очень простыми, для модификации не потребуется смена ядра, переделка всего стека протоколов и не нужны т.п. глобальные обновления.

Полная модификация IPv4 в IPv4r2 потребует создавать полностью новый IPv4r2 стек, как и для IPv6, но реализация функциональности IPv4r2 настолько схожа с IPv4r2, что реально новый IPv4r2 стек это будет копия функций старого IPv4 стека, часть IPv4 функций которого будут слегка, довольно незначительно, модифицированы.

Подсчет контрольной суммы IPv4r2.

IPv4r2 создан в предположении, что логические протоколы вообще не должны обременять себя контролем целостности при передаче данных (за исключением поддержки логических состояний соединения), хотя бы потому, что они не знают какова среда для передачи данных и какие сбойные факторы в этой среде действуют, таким образом физически передавая IPv4r2 пакеты вы должны обертывать его в такую контрольную обертку, сложность которой зависит от проблем с передачей данных в конкретном канале.

В этом плане подсчет контрольной суммы IPv4 это операция достаточно трудоемкая, достаточно ненадежная и в целом архаичная.

IPv4 реализует идею перенести все вычислительные нагрузки по контролю целостности пакетов на источник и адресат пакета, чтобы освободить от этого промежуточные узлы, но такая идея не может сработать в большой сети, состоящей из сегментов с разным качеством и надежностью передачи информации. Тем более что при маршрутизации пакета IPv4 эту контрольную сумму приходится корректировать и в общем говоря пересчитывать.

Заместо такого переноса вычислений, IPv4 фактически работает в предположении, что все участки сети, по которым будут

предаваться данные, должны быть не хуже, чем такие участки, при передаче по которым контроль целостности можно обеспечить подсчетом контрольной суммы как в IPv4. В результате для хороших участков сети это избыточно, а для плохих - недостаточно.

Контроль целостности при передаче данных это задача того уровня сетевого взаимодействия, на котором данные передаются физически. Пакеты IP должны получаться и отправляться на нижележащие сетевые уровни также, как происходит чтение секторов с контроллера диска (каким именно диском управляет контроллер пользователю неизвестно), при этом пользователь, получив сектор от контроллера диска, никакой контрольной суммы не вычисляет.

Например, подсчет контрольных сумм применяется для ПЗУ персонального компьютера и происходит разово, например при начальной загрузке, чтобы убедится что данные для канала связи, сохраненные в ПЗУ, целостные. Это необходимо потому что само ПЗУ исторически было таким, что при хранении данных могло иногда их терять. Если же источник предоставляет для передачи по каналу связи заведомо целостный пакет IPv4, а это требование всегда выполнено, если ОЗУ компьютера исправно, то контрольную сумму в приемнике вычислять нет нужды, качество передачи должно быть обеспечено каналом связи.

С точки зрения уровней сетевого взаимодействия должен быть отдельный от IP протокол, который позволяет накладывать разные алгоритмы контроля целостности при передаче IP пакетов на участке сети, например для выполнения подсчета контрольной суммы IPv4 этот протокол может иметь такой заголовок:
<протокол контроля=1>[8]<версия=1>[8]<контрольная сумма>[16]<размер данных>[32]<любые данные>[размер данных*8]

- размер данных
  размер данных идущих после заголовка в байтах
  не включает в себя размер фиксированного заголовка этого протокола подсчета контрольной суммы (размер заголовка равен 8 байтам)

Поэтому IPv4r2 работая в основном режиме позволяет не использовать подсчет контрольной суммы, если это сделано на нижележащих уровнях.

Арифметика дополнения до единицы в IPv4.

Во многих доступных людям ПК встретить "арифметику дополнения до единицы" (в наших терминах "арифметику обратных кодов") не просто. Поэтому вопрос "что это такое и зачем она нужна" сразу возникает.

Можно сказать, что если суммировать так, что возникающий от двоичного суммирования перенос затем прибавляется в младший разряд суммы, то вся сумма из-за переполнения деградировать в ноль никогда не сможет.

- Второй раз перенос при суммировании результата с битом переноса не возникает. Если самое большое для данного числа бит число, например FF для 8 бит сложить с самим собой (с другим самым большим числом), это все равно что умножить

FF на два или сдвинуть FF влево на один разряд и получить 1FE, в результате самый правый бит даже у самой большой суммы всегда чистый, и если перенос прибавить к младшему разряду, то второй раз переполнение возникнуть не сможет.

- Раз битов хватило для представления суммы самых больших чисел и их переноса, то если складывать числа меньше самых больших, то второй перенос тем более не может возникнуть. Это можно увидеть, заметив что в числе меньшем FF есть хоть один нулевой бит, а значит на этом нуле при суммировании единицы процесс второго переноса и остановится.

Сумма при таком суммировании конечно тоже деградирует, но в некоторое ненулевое значение. Это позволяет резервировать значения суммы 0 для каких-то еще целей, никаких иных особых преимуществ сложение с собственным флагом переноса, помещенным в младший разряд слагаемого, кажется не сулит.

Зато это сулит проблемы от того, что нам надо не только складывать, но и вычитать, чтобы вычислять небольшие изменения в контрольной сумме от поля TTL и т.п., не пересчитывая всю сумму полностью.

Суммируя с переносом мы попадаем в арифметику обратных, а не дополнительных кодов, поэтому чтобы вычислить разность, надо представлять вычитаемое как отрицательное число в обратном коде (командой NOT) с последующим сложением этого отрицательного числа в обратном коде с переносом в младший разряд, это автоматически компенсирует возможное добавление переноса к младшему разряду в предыдущих суммированиях.

Арифметика обратных кодов получается, если условно взять старший бит за знак, а инверсией бит положительные числа менять на отрицательные и наоборот т.е. разбить беззнаковые числа на диапазоны

- 01-7F
- 00
- FF
- FE-80

Инверсией бит мы добиваемся, что сложив положительное и отрицательное числа равные по модулю мы получаем -0 (FF)
В обратном коде основной ноль это -0 (FF), а двоичный ноль +0 (00), получается только при вычитании числа -0 и при константном задании +0 в качестве аргумента.

По кодам видно, что в арифметике обратных кодов прибавление переноса в младший разряд иногда получается когда хотя бы одно из чисел, которое мы суммируем, является в этом коде отрицательным (у одного включен бит в старшем разряде) и всегда получается если складываем два отрицательных числа (у обоих включен бит в старшем разряде).

Код называется обратный, потому что ряду положительных 00, 01, 02 чисел через +1 соответствует ряд отрицательных FF, FE, FD через -1, т.е. числа в отображении как бы выставлены в обратном порядке возрастания, поэтому при такой сортировке и положительные и отрицательные числа отстоят друг от друга на +1, а также оказываются битовой инверсией по отношению друг

к другу

- 00 - FF
- 01 - FE
- ...
- 7F - 80

В обратном коде отрицательные и положительные числа тоже получаются путем перехода на +1 и -1, но можно видеть, что в обратном коде отрицательные числа сдвинуты относительно двоичного 0 (+0) на -1, поэтому арифметика привычного дополнительного кода как N повторов +1 и -1 для них невозможна и после арифметических операций с отрицательными числами, в зависимости от результата операции, может быть необходима коррекция на +1.

Например, складывая в обратном коде -1 и -1, смещенные в численном виде на (-1), имеем

- -1(-1) + -1(-1) = -2(-2) = -4
- реальный результат -2(-1) = -3
- надо прибавить +1

- 1(+0) + -2(-1) = -1(-1)= -2
- реальный результат -1(-1) = -2
- не надо прибавлять +1

- 2(+0) + -1(-1) = 1(-1)= 0
- реальный результат 1(+0) = 1
- надо прибавить +1

Для простоты реализации обратного кода прибавление +1 берется от флага переноса всегда, если он установился после суммирования, даже если с точки зрения арифметики есть переполнение. По этому правилу суммируя в обратном коде (с учетом переноса) и в случае появления переполнения для отрицательных чисел мы добиваемся симметрии для последующего добавление обратного положительного числа и устранения переполнения. Это качество используется и в расчете модификации контрольной суммы IPv4 путем вычитания старого и добавления нового числа, при этом как раз важно, чтобы операции сложения и вычитания одного и того же числа были бы симметричны по отношению к переполнению.

Важное замечание. Хотя в IPv4 заголовках как правило есть хоть одно ненулевое (в значении двоичного кода - хоть один бит не равен 0) слагаемое, в принципе контрольная сумма IPv4 может пересчитываться методом "вычитания старого и прибавления нового значения по правилам обратного кода" только в том случае, если новая сумма содержит хоть одно ненулевое (хоть один бит не равен 0) слагаемое, иначе новая сумма после вычитания и сложения получится равной FFFF, а должна была бы быть 0. С точки зрения обратного кода разницы нет, с точки зрения контрольной суммы разница есть.

Тут есть два пути, или считать что контрольная сумма 0 равна контрольной сумме FFFF, так как этого требует арифметика обратного кода, либо зарезервировать необычное для IPv4 контрольной суммы значение 0 под флаг. В протоколе IPv4r2 значение 0 резервируется под флаг (в IPv4 заголовке хранится битовая инверсия этого значения, т.е. зарезервированный 0 в IPv4 заголовке хранится как FFFF).

Фрагментация IPv4r2.

Если попытаться обойтись без фрагментации IPv4 пакетов, то при обмене данными с помощью IPv4 источник должен ограничить размер пакета самым узким участком всей сети, посылая по широким участкам сети множество мелких пакетов.

Протокол IPv4 предполагает, что фрагментация пакетов это вещь достаточно универсальная, известная на стороне приемника и передатчика, так что при передаче IPv4 пакетов, также как для подсчета контрольной суммы, применяются средства самого IPv4 для обеспечения такой фрагментации.

Значит качество обеспечения универсальной фрагментации в протоколе IPv4 казалось бы могло быть лучше, чем универсальная защита целостности заголовка IPv4 путем подсчета контрольной суммы, но в реальности это не так. Потому что фрагментация силами IPv4 не только усложняет заголовок IPv4 пакета, но также усложняет саму маршрутизацию, которая вынуждена работать с IPv4 фрагментами.

Какие же проблемы при маршрутизации IPv4 фрагментов? Заметим, что по сути при фрагментации на протокол IPv4 возложена задача протокола TCP по передаче и сборке фрагментов, при том что фрагменты при IPv4 фрагментации передаются по правилам UDP. В результате огромный IPv4 исходный пакет может быть утрачен при утрате единственного фрагмента этого пакета, при том что сеть, передающая эти фрагменты, выполнила 99,9% процентов всей работы по передаче этого IPv4 пакета.

Таким образом, опять, как и при подсчете контрольной суммы, оказывается что маршрутизация фрагментов зависит от проблем на конкретных участках сети, о которых IPv4 адресаты ничего не знают и протокол IPv4 не может предоставить средства для правильной фрагментации.

Поэтому IPv4r2 работая в основном режиме предполагает, что фрагментацией, пусть и универсальным образом, должен заниматься отдельный протокол на проблемных участках канала передачи данных, аналогично протоколу для контроля целостности данных, внутрь которого будет вложен IPv4r2 пакет.

Перечислим пару основных путей обеспечения фрагментации IPv4r2, поскольку это в общем менее известно, чем разные способы помехо- защищенного кодирования:

- фрагментация правилами UDP с маршрутизацией (используется в IPv4, применяется для надежных сетей);
- фрагментация правилами UDP без маршрутизации на пути точка-точка (применяется на надежных участках сети);

- фрагментация правилами TCP с маршрутизацией фрагментов (применяется в обычных сетях).

Интерес представляет именно третий вариант, когда узел, вынужденный выполнить маршрутизацию, просто использует TCP протокол, внутри которого вложен IPv4r2 пакет, устанавливая канал связи до того маршрутизатора, который сможет передавать исходный IPv4r2 пакет, в лучшем случае этот маршрутизатор знает о размере своей сети со этим MTU и пробросит исходный IPv4r2 пакет от одного конца своей сети до другого в виде фрагментов, в худшем случае, конечной точкой маршрута станет сам получатель IPv4r2 пакета и начиная с узкого места в сети в адрес получателя IPv4r2 пакета пойдут фрагменты исходного пакета, упакованные в TCP сегменты.

Естественно, что размер IPv4r2 пакета, который может быть передан промежуточным маршрутизатором хотя бы в виде фрагментов, не может быть каким угодно большим. В общем случае IPv4r2 источник должен заранее знать IPv4r2 пакеты какого размера могут быть доставлены до конкретного IPv4r2 назначения по конкретному маршруту.

Для решения вопроса гарантированной, в смысле подходящего размера IPv4r2 пакета, доставки IPv4r2 пакета есть два способа:

- задание максимально размера IPv4r2 пакета пригодного для передачи по любым IPv4r2 сетям (межсетевая работа в интернете);
- поиск максимального размера IPv4r2 пакета для доставки по заданным путям до заданного адресата (работа в специальных и локальных сетях).

Поскольку IPv4r2 это межсетевой протокол, работа в специальных и локальных сетях для него не основная и предполагаемый практический способ установления максимального размера пакета в таких специальных и локальных IPv4r2 сетях, это задание максимального размера IPv4r2 пакета администратором этой сети в ручную.

MTU 1152 в сетях IPv4r2 для межсетевой работы.

Для протокола IPv4 для межсетевой работы установлено требование гарантированной передачи на каждом участке сети без фрагментации IPv4 пакета максимального размера 576 байт (MTU 576), состоящего из:

- 512 байт полезных данных в пакете;
- 64 байт вспомогательных данных управления передачей, включающих в себя заголовки IP/UDP/TCP;

т.е. в IPv4 маршрутизаторах для каждого IPv4 пакета 64 байта максимум отводится на хранение управляющей информации и 512 байт максимум отводится на хранение данных (если 64 байт для хранения управляющей информации не хватает, то что не поместилось в 64 занимает место в блоке из 512 байт, затрудняя обмен данными блоками по 512 байт), в IPv4 на 8 порций пакета идет 1 порция управляющей информации (всего 9 порций по 64 байта), т.е. 1/9=11% управляющей информации.

Пакеты IPv4 большего размера могут потребовать фрагментации и попадают в разряд работы в специальных и локальных IPv4

сетях.

IPv4r2 оперирует б`ольшими адресами, которые не всегда помещаются в 64 байта вместе с опциями и прочей управляющей информацией, поэтому размер передаваемых в IPv4r2 пакете данных имеет все шансы стать менее 512 байт, чтобы передаваться по сетям IPv4 без фрагментации. Поэтому, а также чтобы сохранить и для IPv4r2 соотношение в 11% управляющей информации, максимальный размер пакета IPv4r2, который должен гарантированно передаваться в сетях IPv4r2 без фрагментации, удваивается, т.е. это максимум 1152 байта в пакете (MTU 1152):

- 1024 байт на полезные данные;
- 128 байт на управляющую информацию.

IPv4r2 работая в основном режиме для межсетевой работы использует MTU 1152. Пакеты IPv4r2 большего размера могут потребовать фрагментации и попадают в разряд работы в специальных и локальных IPv4r2 сетях.

Заголовок IPv4r2 с максимальным размером 80 байт.

Расширение адресации IPv4r2 в среднем требует дополнительно 20 байт для заголовка IPv4r2. Чтобы вписаться в 11% на управляющую информацию и в 128 байт, оставив 48 байт на заголовок UDP/TCP, в основном режиме работы IPv4r2 существует расширение размера заголовка IPv4r2 на 20 байт, так что максимальный размер IPv4r2 заголовка станет равен 80 байтам и при этом функциональность IPv4r2 заголовка по опциям будет не хуже чем для IPv4.

Включение режима максимальный размер 80 байт для заголовка IPv4r2, автоматически включает режим максимальный размер 48 байт для:

- заголовка IPv4 протоколов UDP/TCP в стеке протоколов IPv4r2;
- заголовка IPv4r2 протоколов UDPr2/TCPr2.

Превышение размера IPv4r2 заголовка при использовании IPv4 опций контролируется также как и для IPv4, что дает возможность передавать 1024 байт полезных данных без риска IPv4r2 и UDP/TCP заголовкам мешать этой области. При включении опций IPv4r2 для увеличения размера IPv4r2 заголовка на произвольную величину, контроль расхода 128 байт на управляющую информацию лежит на пользователе.

Расширение размера IPv4r2 заголовка до 80 байт предназначено:

- для межсетевой работы IPv4r2 (при работе в интернет), когда имеет значение MTU 1152;
- чтобы давать больше гибкости в использовании опций IPv4.

Расширение размера заголовка IPv4r2 до 80 байт не предназначено:

- для замены сложных протоколов маршрутизации или отладки сети опциями заголовка IPv4, эти опции IPv4 имеют

ограниченное применение для особенных случаев.

Если надо доставлять IPv4r2 заголовок по специальному маршруту или заниматься отладкой сети, то IPv4r2 заголовок должен быть обернут в транспортный или отладочный протокол для такой маршрутизации или отладки, при этом исходный IPv4r2 заголовок, содержащий только конечные адреса маршрута, становится протоколом более высокого уровня.

Фрагментация IPv4r2 пакета флагами IPv4.

Пакеты IPv4r2 с IPv4 флагом запрета фрагментации не должны использовать поля идентификатор и смещение, устанавливая их в 0, это технические поля которые нужны только при фрагментации и которые имеют значение только между теми хостами, которые передают между собой фрагменты. Незнакомые друг с другом конечные IPv4 адресаты точно не могут знать о каких то магических значениях поля идентификатор, а метки потоков и дополнительная адресация задаются отдельными опциями IPv4r2 или протоколами более высокого уровня.

IPv4 флаг запрет фрагментации должен устанавливаться по умолчанию для пакетов размером меньше максимально допустимого MTU, а при межсетевой работе нет никаких причин для протокола более высокого уровня отсылать в неизвестный адрес пакеты б`ольшие чем MTU (б`ольшие чем максимально допустимый для отсылки без фрагментации), т.к. какой бы размер пакета IPv4r2 не был взят, данные в протоколе более высокого уровня все равно придется фрагментировать по этому размеру блока и нет никаких причин, кроме попыток уменьшить нагрузку на сеть устранением лишних заголовков, чтобы не взять этот размер блока сразу корректным для IPv4r2. Таким образом в правильно настроенной IPv4r2 системе межсетевой обмен осуществляется пакетами IPv4r2 меньшими чем MTU 1152 и с включенным флагом запрета фрагментации.

Сетевые протоколы IPv4r2 очень высокого уровня должны работать с блоками полезных данных не более 512 байт, тогда они не будут в нормальном случае генерировать фрагментацию. Поскольку протоколы могут быть вложенными и в UDP/TCP, то IPv4r2 с MTU равным 1152 позволяет оборачивать 512 байт реальных данных довольно большое количество раз.

Для работы именно в сетях IPv4r2 приложение не перегружающее данные заголовками может ориентироваться на блоки полезных данных по 1024 байта с целью уменьшения затрат сети на служебные данные, но работа без задержек в физических сетях с общей средой передачи и множеством пользователей все равно возможна только с малыми пакетами полезных данных по 512 байт.

Малые пакеты нужны потому, что общая скорость передачи при перегрузке в такой сети упадет в любом случае, но для тех приложений, для которых важна скорость реакции в реальном времени (например, по сети передаются события нажатий на кнопки клавиатуры), важно чтобы пусть малые порции данных, но доставлялись бы с минимальной задержкой.

Малая задержка в такой сети возможна только тогда, когда один клиент не занимает общую среду передачи в сети надолго, т.е. если размер передающегося пакета мал. Для малой локальной сети ethernet удовлетворительным считается размер

полезных данных 512 байт, это условная величина, взятая за оптимальную, но при любой взятой заместо 512 величине, при росте относительно нее размера пакета, параметры отклика сети в реальном времени для передачи малых данных реального времени ухудшаются.

Отсылать пакеты IPv4r2 больших размеров можно только в локальных сетях, в специальных глобальных сетях и в локальных соединениях точка-точка, для локальных связей точка-точка IP протокол явно не нужен и используется там только для уменьшения числа используемых протоколов и для однообразия работы со всеми машинами в любых сетях.

Асимметричная адресация IPv4r2 (77 бит).

Главной проблемой исчерпания IPv4 адресов для пользователя (и главной основой для возникновения торговли IPv4 адресами и некоторыми видами хостинга) является то, что пользователи не могут выставлять в сеть серверные ресурсы, поскольку другие люди эти ресурсы не могут адресовать. Сами пользователи могут легко выходить на "крупные" сервера, которые имеют IPv4 адреса, через шлюзы провайдера. Ситуация в чем то аналогичная ADSL, по асимметрии.

Отметим, что работа через шлюз заставляет провайдера создавать вычислительные ресурсы маршрутизатора, которые занимаются трансляцией IPv4 адресов, но опять же, проблема тут только в протоколе IPv4, который всеми был в свое время принят, вот за это и расплата. Теперь от IPv4 взять и просто отказаться уже нельзя.

По большому счету вопрос трансляции IPv4 адресов можно считать у провайдера уже решенным - оборудование для трансляции IPv4 адресов уже установлено. Также вычислительная нагрузка на шлюзы провайдера:

- по маршрутизации IP пакетов без их модификации;
- по физической ретрансляции пакетов в сетях одного типа;
- по необходимости в передаче пакетов между разными физическими типами сетей;

в любом случае останется, даже при IPv6.

Отметим, что некоторым пользователям вообще не понадобится реальный IPv4r2 адрес для предоставления доступа к своим компьютерам из сети, потому что у них нет серверных ресурсов.

Пока провайдеры имеют ресурсы обеспечивать клиента динамическими IPv4 адресами, нам выгодно рассмотреть асимметричную схему адресации IPv4r2 сетей, потому что:

- размер заголовка IPv4 ограничен и много- битовая адресация уменьшает функциональность IPv4 сетей;
- много- битовая адресация всегда ведет к росту накладных расходов при передаче полезных данных малыми пакетами;
- нам для установления связи надо уметь адресовать только сервера в глобальных IPv4r2 сетях.

Равноправное общение двух IPv4r2 адресов при асимметричной адресации происходит по "кольцевой" схеме с участием IPv4 шлюза провайдера:

- исходящие двух- или однонаправленные соединения с первого IPv4r2 адреса идут через динамический IPv4 его (первого) провайдера на полный IPv4r2 адрес второго IPv4r2;
- при необходимости установить ответное исходящее соединение, второй IPv4r2 адрес через динамический IPv4 уже своего (второго) провайдера обращается на полный IPv4r2 адрес первого IPv4r2.

Расширенная адресация IPv4r2 для шлюзов (индексы локальных сокетов).

Протоколы UDP/TCP имеют собственную, независимую от IP, адресацию, которая называется портами (адрес каждого порта это число, как адрес байта памяти, например, порт 100). По сути порт UDP/TCP это аналог аппаратного порта ввода-вывода для архитектур компьютеров времени создания UDP/TCP.

В порт ввода-вывода можно писать данные или читать из него, а то что будет при этом происходить зависит от того, как эти порты соединены физическими проводами снаружи компьютера между собой или с другими компьютерами и прочими внешними устройствами.

Порты UDP/TCP делают то же самое, если их заместо физических проводов программно соединить с другими портами. Когда такое соединение сделано, и по протоколу UDP/TCP предаются данные, можно говорить о канале передачи данных типа точка-точка.

Чтобы соединить между собой порты UDP/TCP на разных компьютерах, каждый такой компьютер должен иметь свой сетевой адрес, например, IPv4, NetBIOS и т.д. Один или более адресов от одного или более протокола, нужных совместно для осуществления успешной связи, в данном случае это будет "сетевой адрес:порт UDP/TCP", называется сокетом. Пусть такие соединяемые компьютеры имеют IPv4 адреса, тогда сокет это пара адресов "IPv4:порт".

Каждый UDP/TCP канал передачи данных, имеющий тип точка-точка, можно описать парой сокетов на концах такого соединения: источник и назначение.

При трансляции IPv4 адресов шлюзом провайдера, каждому реальному IPv4 адресу и порту шлюза ставится в соответствие локальный IPv4 адрес и порт источника и IPv4 адрес и порт назначения.

источник- локальный IPv4: локальный порт
шлюз провайдера- реальный IPv4: реальный порт
назначение- назначение IPv4: назначение порт

Т.е. для компьютера назначения все выглядит так, словно с ним работает шлюз от имени своего реального (внешнего) IPv4 адреса и порта, хотя на самом деле шлюз отправляет все приходящие внешние данные на локальный IPv4 адрес и порт источника и в обратную сторону.

В терминах сокетов, каждому сокету шлюза ставится в соответствие UDP/TCP канал передачи данных
 сокет шлюза:
  UDP/TCP канал передачи данных:
   сокет источника
   сокет назначения

Чтобы шлюз мог отправить обратный IPv4 пакет от "назначение IPv4: назначение порт", который приходит на "реальный IPv4: реальный порт", в правильный "локальный IPv4: локальный порт", шлюз должен понять кому в локальной сети шлюза на деле адресован пакет, приходящий на "реальный IPv4: реальный порт" шлюза. Сокет шлюза должен определить какой UDP/TCP канал передачи данных используется этим IPv4 пакетом, пришедшим от сокета назначения.

Шлюз в малой локальной сети.

а) Это можно решить так, как делается в малой локальной сети:
каждому "локальный IPv4: локальный порт" шлюз динамически выделяет отдельный реальный порт шлюза по запросу от источника, т.е. с каждым "реальный IPv4: реальный порт" шлюза связан только один:
"локальный IPv4: локальный порт"

Этим на локальной половине исходного канала "сокет источника - сокет назначения" создается уникальная связь "сокет источника - порт шлюза", так что внешние сокеты не могут адресовать, т.е. физически не могут указать какие-либо иные порты локального источника, кроме того что связан с этим сокетом источника. Сокет же источника может адресовать любой сокет назначения через этот порт шлюза.

Тогда если ответные данные пришли на этот "реальный IPv4: реальный порт" от любого внешнего адреса, они могут быть правильно отправлены на "локальный IPv4: локальный порт". И наоборот, данные пришедшие от "локальный IPv4: локальный порт" могут быть правильно отправлены на любой внешний сокет.

Проблема тут в том, что адресация UDP/TCP портов всего лишь 16 битная. Для одной машины этого может и хватает, но вот для шлюза провайдера, обслуживающего всех его клиентов, этого явно мало, поскольку клиентов больше, чем 60 тысяч.

На самом деле даже один клиент может открыть десятки, сотни исходящих портов одновременно и для каждого из них шлюз провайдера должен выделить отдельный уникальный порт шлюза, т.е. одним IPv4 адресом провайдера такого шлюза реально сможет пользоваться только около тысячи обычных клиентов одновременно, что хватает для малых локальных сетей, не не подходит для провайдера.

б) Таким образом, для обеспечения лучшей работы IPv4 шлюзов, в том числе защитных экранов для локальных сетей, нужно

расширение адресации сокетов, а поскольку сетевой шлюз предполагает единственный сетевой адрес (IPv4 ли это адрес или IPv6 адрес - это неважно), то расширять можно только адреса портов.

Есть два пути расширения адресации портов:
- расширять адрес порта UDP/TCP, превратив их из 16 битных в 24 или 32 битные;
- добавить индексы локального сокета шлюза для сетевого адреса IPv4, специально чтобы поддержать шлюзы.

Индексы локального сокета шлюза.

Добавить индексы шлюза для сетевого адреса IPv4 выгодно потому, что тогда обычный шлюз сможет работать только на сетевом уровне IPv4, независимо от типа протокола UDP/TCP или иного протокола, что и сделано в протоколе IPv4r2. Поэтому получается IPv4r2 сокет такого вида: "сетевой адрес IPv4r2:индекс локального сокета шлюза".

Теперь IPv4r2 шлюз каждому клиенту с сокетом "локальный IPv4: локальный индекс шлюза" ставит в соответствие сокет шлюза "реальный IPv4: реальный индекс шлюза", независимо от того какой протокол и какие порты используются внутри IPv4r2 пакета.

Для того чтобы это сработало и шлюз и назначение должны понимать что такое "индекс локального сокета шлюза". Если IPv4r2 сервер назначения не понимает "индекс локального сокета шлюза", то шлюз должен работать с таким назначением по старой схеме, с анализом портов UDP/TCP, поэтому если шлюз не анализирует протоколы UDP/TCP, то может отвергать пакеты с индексом 0 приходящие от IPv4r2 назначения на IPv4 шлюза, поскольку такой шлюз никогда не назначает своих локальных клиентов на индекс 0 (под таким индексом работает любой клиент и сам шлюз как клиент, не перенаправляя данные от своего локального клиента).

Индекс сокета шлюза не нужен для адресации пользователем и не входит в URL запроса к серверу, он создается динамически на шлюзах и используется только IPv4r2 сервером и IPv4r2 шлюзом. Теоретически локальный компьютер-источник может (в URL IPv4r2 его можно указать так "протокол://IPv4r2 адрес:UDP/TCP порт:IP индекс"), но должен воздерживаться от того, чтобы использовать индексы сокета шлюза (всегда используя индекс сокета 0). Перегруженный шлюз вправе отбросить пакеты от клиента с индексом локального сокета не равным 0, если по мнению шлюза этот клиент сам не может быть шлюзом.

Шлюз работающий на сетевом уровне:

- может назначать индексы сокета шлюза своим клиентам динамически при каждой новой авторизации пользователя или первом после тайм- аута исходящем от этого клиента IPv4 пакете (при этом на практике для достаточно непрерывно посылающего пакеты клиента будет создан постоянный на каждый непрерывный сеанс авторизации IPv4r2 канал, с постоянным индексом сокета шлюза связанным с локальным IPv4 этого клиента);
- может разделять один индекс сокета шлюза между несколькими клиентами, отсылая входящие на этот индекс сокета пакеты всем клиентам, для которых этот индекс сокета назначен;

- может проверять, что адрес назначения тот, который был до истечения тайм- аута указан в исходящих запросах клиента.

Индекс сокета шлюза, работающего на сетевом уровне, это просто расширение IPv4 адресации, которое позволяет подключать больше компьютеров локальной сети к одному IPv4 адресу, но это такое малое расширение IPv4 адресации, которое в общем требует меньше места в IPv4 заголовке для хранения расширения, например 16 битные индексы займут в IPv4 заголовке минимум 2 и максимум 4 байта и то не на всех участках сетевого маршрута. Размер индекса произвольный, разумные значения: 16, 32, 48 бит на индекс, что потребует в IPv4 заголовке максимум 8 байт.

Также индекс сокета позволяет шлюзу работать в режиме IPv4 моста, позволяя через индекс сокета адресовать все UDP/TCP порты локального компьютера связанного с этим сокетом.

Индекс сокета шлюза, работающего на уровне UDP/TCP по схеме (а) шлюза в малой локальной сети, эквивалентен расширению адресации UDP/TCP портов. Поскольку это расширение адресации UDP/TCP будет прозрачно для UDP/TCP пользователей, не затрагивает локальные машины и динамическое (зависит от загрузки шлюза), это более гибкая система расширения адресации UDP/TCP, если шлюз необходим, поскольку шлюзы служат не только для выхода в корневую IPv4 сеть из локальных сетей, но и для защитной фильтрации трафика, не допуская:
- входящие соединения на компьютеры локальной сети;
- входящие соединения с недопустимых адресов на компьютеры локальной сети;
- входящие пакеты с иных адресов назначения, не указанных в исходящем запросе UDP/TCP от компьютера локальной сети;
- и т.д.

Для сервера реализующего UDP/TCP в стеке протоколов IPv4r2, сокет IPv4r2 имеет три поля:
- сетевой сокет
  - сетевой адрес
  - индекс локального сокета шлюза
- порт UDP/TCP

Индекс сокета шлюза это динамическое присвоение IPv4 клиенту временного IPv4r2 адреса.

Флаг принадлежности индекса шлюза IPv4r2 адресату.

Изначально, при создании протокола IPv4r2, индекс шлюза был предназначен только для работы в глобальной сети IPv4r2 через IPv4 шлюз провайдера, когда адрес клиента всегда IPv4, а адрес сервера всегда IPv4r2, поэтому когда адрес клиента уже IPv4r2, индекс шлюза в принципе не нужен.

Предполагались очень простые правила использования индекса во время обмена при любой комбинации типов адресов одинаковы

- индекс применяется только для клиента, сервер не может иметь индекс шлюза.

Если для асимметричной IPv4r2 адресации использование индекса шлюза не вызывало особых трудностей (предполагалось что шлюз всегда принадлежит IPv4 адресу), то при симметричной IPv4r2 адресации возникают неоднозначности, только глядя на IPv4r2 пакет невозможно понять к какому адресату относится индекс шлюза, если неизвестно кто из адресатов сервер.

Такая неоднозначность недопустима для сетевого протокола, а поскольку для IPv4r2 адресатов индекс шлюза внеплановым образом оказался полезным также, как и для IPv4 адресатов, отказаться от индекса для IPv4r2 адреса нельзя, а чтобы устранить неоднозначности, особенно при симметричной IPv4r2 адресации, старший бит индекса шлюза всегда является флагом:

- 0 шлюз принадлежит адресу источника;
- 1 шлюз принадлежит адресу назначения.

При работе в локальных сетях, на каждом участке локальной сети клиент-шлюз индексируется последний шлюз, который передал пакет на этот участок, каждый IPv4r2 шлюз создает сокет трансляции адреса, локальная часть которого может включать индекс предыдущего шлюза, также как и глобальная часть может содержать собственный индекс.

Плановый для IPv4r2 вариант использования индекса шлюза, это когда индекс шлюза имеет только IPv4 шлюз провайдера имеющий выход в IPv4r2 глобальную адресацию, а остальные шлюзы локальной сети (если они вообще есть и выполняют трансляцию) индексы не используют, т.е. индекс шлюза динамически создает IPv4r2 адрес для IPv4 клиента на глобальном уровне адресации.

Как только шлюз получил флаг адреса, так возникла возможность отказаться от проектной схемы IPv4r2, в которой только один адресат имеет индекс, например, два IPv4 клиента могут общаться между собой на глобальном IPv4r2 уровне через свои шлюзы провайдера, так что у каждого из них будет индекс шлюза. Теперь оба IPv4r2 адресата могут иметь свой индекс шлюза.

Дополнительная IPv6 адресация в сетях IPv4r2.

IPv6 адресация в сетях IPv4r2 применяется тогда, когда нужно доставить IPv4r2 пакет по IPv4r2 сети тому адресату в IPv4r2 сети, для которого известен его IPv6 адрес, а его IPv4r2 адрес не известен.

Это полностью аналогично работе пользовательского адресного пространства, но в отличие от пользовательского адреса, который выделяет сам владелец IPv4r2 адреса, IPv6 адрес выделяется иным образом (по правилам протокола IPv6).

Также использование 128 битных расширений адресов от IPv6, заместо 45 битных расширений адресов Ipv4r2, быстро приводит к исчерпанию свободного места в IPv4 заголовке, но при работе через IPv4 шлюз провайдера это вполне применимо.

При IPv6 адресации IPv4r2 адрес назначения может быть как IPv4, так и IPv4r2; но заместо указания реального адреса

назначения, при IPv6 адресации этот IPv4r2 адрес назначения всегда указывает на IPv4r2 шлюз назначения, который знает как доставить этот IPv4r2 пакет своему IPv6 адресату. Этот же IPv4r2 шлюз работает как шлюз провайдера, т.е. он создает IPv4r2 канал между IPv6 адресатом и IPv4r2 источником по которому можно передавать IPv4r2 пакеты в сторону IPv6 сервера. То же самое в обратную сторону, когда IPv6 сервер отправляет обратные пакеты через IPv4r2 шлюз IPv4r2 источнику этого сеанса связи и создания канала.

Несмотря на кажущуюся сложность, предполагается что этот механизм будет использоваться шлюзом провайдера, который будет выступать в качестве того шлюза, который знает как отправить IPv4r2 пакет тому IPv4r2 адресату, для которого известен его IPv6 адрес.

Важно, что несмотря на свой IPv6 адрес, в итоге этот IPv6 адресат получит IPv4r2 пакет, а не IPv6 пакет (также как и в случае пользовательского расширения адресного пространства).

Как и в случае упаковки IPv4r2 пакета в UDP пакеты IPv4, IPv4r2 пакеты для IPv6 адресата можно отправлять на IPv4 адрес шлюза провайдера, который подчиняется следующим правилам:

- IPv4 адрес шлюза провайдера это тот адрес, что получает ваш маршрутизатор при установлении связи с провайдером в качестве адреса своего шлюза по умолчанию.
- Если провайдер не может обеспечить фиксированный адрес такого шлюза, то заместо адреса шлюза провайдера в локальной сети в качестве шлюза IPv6 можно использовать фиксированный IPv4 адрес 192.88.99.1 (также этот фиксированный IPv4 адрес используется для инкапсуляции протокола IPv6), IPv4 пакеты на этот адрес обычно должны быть перехвачены шлюзом провайдера.

Адресация подсетей IPv4r2.

В наследство от IPv4 протоколу IPv4r2 достались:

- подсети вида "IPv4 адрес нулевого хоста подсети"/"число бит маски подсети", так что в нулевых битах маски сети у IPv4 адреса могут быть любые комбинации битов, определяющие уникальные адреса хостов этой подсети, а применение маски сети к любому IPv4 адресу подсети дает "IPv4 адрес нулевого хоста подсети";
- широковещательные IP запросы к такой подсети на выделенный IPv4 адрес, у которого все биты в поле хоста подсети установлены в 1 (это битовая инверсия маски подсети);
- назначение сетевому интерфейсу работы быть шлюзом до хостов этой подсети, путем указания выделенного IPv4 адреса, у которого все биты в поле хоста подсети установлены в 0 (это маска подсети примененная к IPv4 адресу любого хоста подсети).

Выделение нескольких IPv4 адресов на эти служебные цели создает некоторые трудности, как при формировании малых IPv4 подсетей и IPv4 подсетей сложной структуры, так и с уникальностью адресации хостов в подсетях.

Поэтому в сетях IPv4r2 подсети могут формироваться иным путем, для этого вместо выделенного адреса применяются IPv4r2 опции, которые кодируют:

- число бит маски подсети для IPv4r2 адресата;
- широковещательный запрос для IPv4r2 адреса назначения;

а для обозначения назначения сетевому интерфейсу быть шлюзом до хостов подсети используется запись "IPv4r2 адрес нулевого хоста подсети"/:"число бит маски подсети".

Таким образом, в сетях IPv4r2, нулевой и последний IPv4r2 адрес такой подсети может быть использован как адрес хоста. Также один IPv4r2 адрес хоста может принадлежать одновременно разным IPv4r2 подсетям, но этот IPv4r2 адрес в обычном случае должен принадлежать только одному интерфейсу, т.е. два разных интерфейса не могут отличаться только маской подсети.

Из соображений эффективности кодирования маска сети может задаваться не как число единиц от начала, а как число нулей от конца - обратная маска, обозначение: /:-"число бит обратной маски подсети".

В сетях IPv4r2 указание маски подсети в заголовке IPv4r2 обычно требуется только при отправке широковещательных IPv4r2 запросов в эту подсеть.

Протокол ICMPr2.

ICMP протокол, в версии IPv4 возвращающий первые 64 байта IP пакета (макс заголовок) вызвавшего ICMP ответ, в основном режиме IPv4r2 должен возвращать первые 128 байт всегда, а в режиме совместимости с IPv4 возвращать эти 128 байт по возможности (т.е. если после IPv4 заголовка в ICMP пакете не идет информация ICMP для IPv4).

Работа IPv4 приложений в IPv4r2 сетях.

Для того чтобы IPv4 приложения, которых уже много и которые еще даже будут создаваться новые, могли бы работать с любой иной сетевой адресацией (IPv4r2, IPv6 и т.п. - не важно), можно отображать адреса этой другой сети на адреса протокола IPv4 - делать NAT трансляцию.

В общем такое отображение совершенно обычное решение для IPv4, например, при работе в сети ethernet компьютеры в этой сети имеют не IPv4 адреса, а ethernet адреса (MAC адреса), которые присвоены сетевым картам, и связь между IPv4 и ethernet адресами выполняет IPv4 протокол ARP. В протоколе ethernet тоже есть и адресация, и широковещательные запросы, и даже VLAN подсети.

Поэтому отображение адресов IPv4r2 на IPv4 это тоже совершенно обычное решение для многоуровневой системы сетевых

протоколов.

К использованию IPv4 диапазона "169.254.0.0/16" .

Этот сетевой диапазон 169.254.0.0/16 отведен в протоколе IPv4 для "самой локальной сети из всех локальных сетей", т.е. для сети находящейся в пределах строго одного сегмента (там же работает IPv4 протокол ARP). Это может использоваться в том случае, когда многоуровневая сеть создается несколькими локальными сегментами (например из диапазона 192.168.0.0/16).

Практически просто так назначать фиксированные адреса локальных машин на этот диапазон 169.254.0.0/16 неудобно, поскольку это накладывает обязанность не использовать в такой сети маршрутизаторы, а такого ограничения в малой локальной сети как правило нет. Поэтому диапазон 169.254.0.0/16 остается обычно неиспользуемым.

Также этот сетевой диапазон 169.254.0.0/16 отведен для адресов, которые назначаются машинам локальной сети ethernet в случае, если отказал централизованный сервис DHCP локальной сети (возможно с помощью специального децентрализованного протокола для устранения коллизий одинаковых IP адресов для разных ethernet адресов). В случае если в сети не работает DHCP или такой протокол его замещающий, этот диапазон 169.254.0.0/16 также не используется.

"Локальней" чем сеть 169.254.0.0/16 будет только сеть 127.0.0.0/8, которая вообще ограничена рамками локальной машины и пакеты этой сети в сеть не выходят.

Однако, по причине работы защитных средств и сетевых экранов, которые по умолчанию подразумевают адреса 127.0.0.0/8 для конкретных целей, любую трансляцию NAT неудобно назначать на сеть 127.0.0.0/8, это угрожает неправильной работой этих средств в их установках по умолчанию, так что об отказе в их работе вы даже не будете знать.

Несмотря на использование сети 169.254.0.0/16 для поддержки отказоустойчивости DHCP, в общем это также и обычная локальная сеть, пригодная для NAT, поэтому в каждом конкретном случае можно выбрать какая именно локальная сеть IPv4 будет зарезервирована для IPv4r2 трансляции NAT.

В тестовой сети, где проводятся IPv4r2 испытания, протокол DHCP не используется и сеть 169.254.0.0/16 прекрасно подходит для роли NAT трансляции.

В некоторых случаях для IPv4r2 трансляции можно будет использовать даже диапазон 127.0.0.0/8.

Коллизий с протоколом децентрализованного DHCP в сети 169.254.0.0/16 также можно избежать, поскольку такой протокол никогда не будет работать в локальном сегменте сети из 65 тысяч машин.

Например, изощренный протокол децентрализованного DHCP назначения адресов (речь идет о малых локальных сетях в десяток, несколько десятков, в исключительном случае пару сотен машин) может выделять для себя под- диапазоны из сети 169.254.0.0/16, начиная сверху, от 254:

- 169.254.254.0/24 первые 256 машин;
- 169.254.253.0/24 вторые 256 машин (тут точно уже лучше починить DHCP);
- и т.д.;

а протокол IPv4r2 трансляции выделять для себя под- диапазоны из сети 169.254.0.0/16, начиная снизу, от 11:

- 169.254.11.0/24 первый IPv4r2 адрес NAT трансляции;
- и т.д.;

при этом адреса 169.254.255.0/24 - 169.254.255.254/24, 169.254.0.1/24 - 169.254.10.255/24 доступны для фиксированного задания адресов в локальной сети.

Инкапсуляция IPv4r2 пакетов в UDP поток IPv4.

Такая инкапсуляция разработана для работы на участке сети клиент-маршрутизатор-провайдер, от клиента в локальной сети до шлюза провайдера, чтобы работать в IPv4r2 сети со старыми IPv4 маршрутизаторами.

Несмотря на то, что для того чтобы в любой, несовместимый с IPv4r2, маршрутизатор IPv4 вставить поддержку опций IPv4r2, нужно выполнить элементарные действия, для IPv4r2 маршрутизатора на базе minix это всего лишь замена кода:

```
      default:
      return EINVAL;
```

на код:

```
      default:
      if (opt[1] < 2) return EINVAL;
      i += opt[1];
      opt += opt[1];
```

но на практике, если маршрутизатор локальной сети сделан по принципу "фирмваре", то это значит что никто даже такую замену в прошивку маршрутизатора делать не будет, и значит совместимости с IPv4r2 у такого IPv4 маршрутизатора не будет по причине особой интерпретации этим маршрутизатором протокола IPv4 (а именно особой интерпретации формата опций IPv4), но задачей IPv4r2 является работа и с такими IPv4 маршрутизаторами, существующими на практике.

Тут речь не идет о недостатках протокола IPv4r2, проблемой является обеспечение работы некоторых старых IPv4 системных программ, которые просто написаны некорректно с точки зрения исходного IPv4, они даже с самим IPv4 не работают.

Инкапсуляция в UDP позволяет обеспечить работу маршрутизатора в режиме IPv4 маршрутизатора и работу с IPv4r2 расширениями в промежуточном режиме IPv4r2 моста-маршрутизатора.

Для этого используется инкапсуляция IPv4r2 пакетов в UDP поток IPv4 между любой локальной машиной в локальной сети и шлюзом провайдера, т.е. шлюзу провайдера отправляется UDP пакет, в котором получателем указаны именно IPv4 адрес и UDP порт шлюза провайдера, а не кто-то еще в сети.

Заголовок блока данных UDP инкапсуляции IPv4r2:

- <сигнатура>[32]
  байты 0x55, 0xaa, 0x42, 0x00

после этого идет полный IPv4r2 пакет (первым идет байт начала заголовка IPv4r2 пакета 0x4?).

Если UDP порт сервиса такой инкапсуляции принял UDP пакет, для которого сигнатура заголовка UDP инкапсуляции совпадает, то данные, начиная со следующего после нее байта 0x4? и включая его, передаются IPv4r2 подсистему на этой машине так, как если бы они были приняты сетевой картой по протоколу IPv4r2. И в обратную сторону, IPv4r2 пакеты которые надо отправлять через UDP инкапсуляцию префиксируются заголовком UDP инкапсуляции и отправляются IPv4 адресату, у которого работает сервис UDP инкапсуляции.

Пакеты UDP инкапсуляции отправляются компьютером через IPv4 маршрутизатор как обычные исходящие запросы в интернет через шлюз по умолчанию, но заместо интернета эти пакеты перехватываются шлюзом провайдера. После установления UDP канала со шлюзом провайдера этот компьютер сможет получать и входящие UDP пакеты от шлюза провайдера (режим моста). Пакеты из интернета в общем случае не могут инициировать такой UDP мост через IPv4 маршрутизатор.

Размер IPv4r2 пакета при UDP инкапсуляции вырастет на:

- 20 байт заголовок IPv4;
- 8 байт заголовок UDP;
- 4 байт сигнатура блока данных UDP;

итого 32 байта.

Поскольку MTU IPv4r2 для работы в интернет равен 1152, то добавление 32 байт при UDP инкапсуляции делает макс размер IPv4r2 пакета в 1184 байта и то только на участке сети клиент-маршрутизатор-провайдер, и если этот участок обслуживается сетью ethernet, где размер MTU ethernet равен 1500 байт, и часть из этих 1500, пусть до 64 байт, может быть зарезервирована провайдером для работы каналов типа точка-точка поверх ethernet, но даже в таком случае остается резерв в 252 байта, который позволяет IPv4r2 пакету с MTU 1152 проходить о сети ethernet до шлюза провайдера.

Для того чтобы обеспечить работу IPv4r2 в сетях с фирмваре- маршрутизаторами, шлюз провайдера должен выделить ресурсы для создания UDP канала с локальным хостом клиента, а локальный хост клиента запустить программу трансляции. Также клиент провайдера должен знать IPv4 адрес и UDP порт шлюза провайдера на котором работает такой сервис (по какому адресу посылать UDP пакеты для шлюза провайдера):

- Номер UDP порта провайдера для такого сервиса может быть получен от провайдера или может использоваться фиксированный номер UDP порта 4 (этот UDP порт официально не используется), IPv4 пакеты на этот порт должны быть перехвачены IPv4r2 шлюзом провайдера.
- IPv4 адрес шлюза провайдера это тот адрес, что получает ваш маршрутизатор при установлении связи с провайдером в качестве адреса своего шлюза по умолчанию. Если провайдер не может обеспечить фиксированный адрес такого шлюза, то заместо адреса шлюза провайдера для такой UDP инкапсуляции можно использовать фиксированный IPv4 адрес 192.88.99.1 (также этот IPv4 адрес используется для инкапсуляции протокола IPv6), IPv4 пакеты на этот адрес должны быть перехвачены IPv4r2 шлюзом провайдера.

Контрольная сумма UDP при приеме не используется независимо от значения поля, в исходящих пакетах рекомендуется устанавливать значение 0. Номер UDP порта источника можно установить такой же, как номер UDP порта назначения или любой постоянный, кроме 0, ненулевой порт необходим чтобы маршрутизатор смог доставлять входящие UDP пакеты обратно. Реально шлюз провайдера (или прямо хост назначения) будет отсылать пакеты обратно на тот порт UDP, который будет выделен IPv4 маршрутизатором локальной сети при трансляции адресов исходящих UDP запросов.

Программа UDP трансляции IPv4r2 не создает UDP сессии, заместо этого анализируется каждый входящий UDP пакет адресованный этому сетевому интерфейсу или адресу 192.88.99.1 на предмет сигнатуры "UDP инкапсуляции IPv4r2" после UDP заголовка, т.е. UDP используется как "IP с большим заголовком". Фрагментация такого UDP пакета должна быть запрещена при создании исходящего пакета.

Для UDP потока IPv4r2 инкапсуляции IPv4 маршрутизатор работает в режим моста, т.е. функции маршрутизатора не выполняет, поэтому на стороне клиента такую функцию сетевого фильтра должны выполнять программы самого клиента: на клиенте должна работать системная программа эмулирующая маршрутизатор и не просто передавать входящие IPv4 пакеты, а контролировать, что входящие IPv4r2 пакеты имеют открытые этим локальным компьютером сокеты и выполнять для правильных пакетов NAT трансляцию.

В режиме UDP инкапсуляции, если включена NAT трансляция IPv4 в IPv4r2, то IPv4 пакеты могут сначала проходить через NAT транслятор для формирования IPv4r2 пакета, но такая трансляция адресов NAT будет не просто посылать IPv4r2 пакеты с подменой адресов, как было без использования UDP, но еще и упаковывать их в UDP пакет.

Дополнительное описание протокола "IPv4r2.0".

Для того чтобы IPv4 приложения могли использовать глобальные IPv4r2 адреса, IPv4r2 адреса отображаются на локальные IPv4 адреса, например по умолчанию диапазона 169.254.0.0/16. При этом сколько бы диапазонов IPv4 сети мы не взяли для такого отображения, даже всей адресации IPv4 не хватит чтобы представить все IPv4r2 адреса с помощью IPv4.

При работе в интернет для клиента проблем при доступе к IPv4r2 серверу это не создает, т.к. для клиента продолжает

работать IPv4 шлюз провайдера с глобальным IPv4 адресом, но IPv4r2 сервер с IPv4 серверным приложением не может принимать сообщения от любого IPv4r2 адреса клиента, т.к. в общем случае IPv4r2 адрес клиента невозможно легко отобразить на IPv4 адресацию сервера - не хватит IPv4 локальных адресов.

Также при работе в интернет IPv4 серверное приложение на IPv4r2 сервере не знает свой внешний IPv4 адрес, поскольку этого глобального IPv4 адреса просто не существует (для IPv4 все наоборот - клиент не знает свой внешний IPv4, но который все же есть в конечном итоге у провайдера) и не может никому указать такой IPv4 глобальный адрес для доступа к самому себе через интернет.

Поэтому те устаревшие реализации IPv4 приложений, которые по своему дизайну жестко привязаны к IPv4 адресации и внутри своих пакетов данных даже обмениваются IPv4 адресами (а не берут эти адреса из IPv4 заголовка), не смогут работать на IPv4r2 сервере без специальных модификаций как с серверной, так и с клиентской стороны. Пример такого проблемного IPv4 протокола для работы в IPv4r2.0 это FTP (см. пример работы FTP в сети IPv4r2.0 далее).

Со стороны клиента нет проблем чтобы отобразить каждый желаемый IPv4r2 адрес сервера на свой IPv4 адрес - одновременно свободных IPv4 адресов в адресном пространстве IPv4 клиента десятки тысяч и за время одной практической сессии пользователю не требуется обращаться одновременно к такому большому количеству внешних серверов. Реально нет и тысячи разных внешних IPv4 серверов с которыми одновременно идет работа пользователя, т.е. если пользователю хватает для этого IPv4 адресов при работе в интернет сейчас, то добавление отображений IPv4r2 адресов на IPv4 заметно не изменит эту ситуацию.

При работе в специальной и локальной сети, и сервер, и клиент, которые знают друг друга, оба могут иметь при обмене IPv4r2 адреса и для этих IPv4r2 должны быть постоянно заданы отображения на свои IPv4 как у клиента, так и у сервера.

Также и при работе в интернет пользователь может задать постоянное отображение интересных ему внешних IPv4r2 серверов на свои IPv4 локальные адреса и обращаться к ним через IPv4 шлюз провайдера.

Если у клиента не работает IPv4r2 DNS клиент (или отдельное приложение для работы с IPv4r2.0), которые могут динамически (по запросу) преобразовывать новые в этой сессии IPv4r2 адреса в их отображения на IPv4 и вести кэш таких отображений, то статическое задание IPv4r2 отображения на IPv4 это единственный способ доступа к IPv4r2 адресам в интернет, который напоминает межсетевую работу в IPv4 сети P2P.

Поскольку пока IPv4r2 серверов относительно мало, пока динамическое задание IPv4r2 отображения на IPv4 практически не используется и пока нужные IPv4r2 адреса серверов можно задавать статически даже для работы в интернет; а с учетом IPv4r2.0 предназначения, такое статическое задание отображений практически может использоваться очень длительное время после введения IPv4r2.0, поскольку серверных ресурсов пользователя относительно мало, как и в IPv4 сетях P2P.

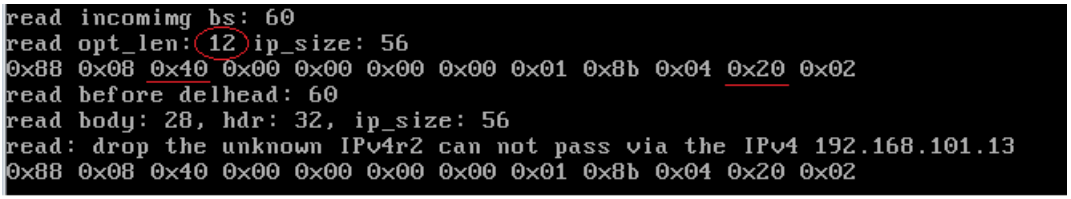
Стандарт IPv4r2.0 не допускает применения индекса шлюза, потому что индекс шлюза по своему назначению генерируется на шлюзе:

- для источника, т.е. статическая запись о IPv4r2 адресе назначения обычно не может иметь индекс шлюза;
- динамически, т.е. статическая запись о IPv4r2 адресе источника обычно не может иметь индекс шлюза;
- в больших количествах, т.е. по размерности адресное пространство индексов немедленно переполняет возможности IPv4 адресации.

По сути индекса шлюза нужен для того, чтобы избавиться от необходимости иметь глобальные IPv4 адреса источников, но не задавая для них фиксированные IPv4r2 адреса, в то время как IPv4r2.0 стремится к обратному - к работе с глобальными IPv4 адресами источника заместо IPv4r2 адресов (для совместимости с IPv4 серверными приложениями).

Работа IPv4 FTP в сети "IPv4r2.0".

```
Работа minix реализации FTP IPv4r2 сервера и IPv4r2 клиента
при соединении с t3r2(169.254.0.131:21) от t1r2(169.254.0.111:32772)
# ftp
ftp>passive
Passive mode is now ON
ftp>open t3r2
220 FTP service (Ftpd 2.00) ready on mx3 at Thu, 18 Feb 2016 16:17:49 GMT
ftp>status
211-mx3(192.168.101.13:21) FTP server status:
    Version 2.00  Thu, 18 Feb 2016 16:17:55 GMT
    Connected to 169.254.0.111:32772
    Not logged in
    MODE: Stream
    TYPE: Ascii
211 End of status
...
ftp>ls
227 Entering Passive Mode (192,168,101,13,128,5).
...
```



Это ftp источник t1r2 ошибочно генерирует запрос к 13 заместо 131

Как видно ftp сервер minix оперирует своим локальным 192.168.101.13 и ftp клиент minix при передаче данных, правильно указывая свой IPv4r2 адрес, пытается подключиться к локальному IPv4 адресу 13 сервера, несмотря на исходный запрос клиента к внешнему IPv4r2 адресу сервера (отображенного на 169.254.0.131 IPv4 клиента).

Без модификации ftp сервер и клиент minix не будут работать даже с оригинальным IPv4 через маршрутизатор сети, т.к. ftp сервер требует уникальных на глобальном уровне IPv4 адресов или отдельной от интернет локальной сети, что по нынешним временам не может считаться приемлемой реализацией ftp. В реализации такого поведения протокола ftp все жестко связано с 32 битной глобальной адресацией IPv4, так что такая реализация не пригодна для IPv4r2.

Надо признать, что ftp, разработанный на заре IPv4, это один из особых случаев IPv4, этот протокол по умолчанию не работает не только с маршрутизатором IPv4 локальной сети, но и в некоторых системах IPv4 маршрутизации даже с собственным локальным адресом 127.0.0.1, который находится на собственной машине, а для современной работы IPv4 пользователя по ftp в 99% практических случаев нужно задействовать режим PASSIVE, поскольку ftp клиент не имеет IPv4 адреса.

Не вдаваясь в детали ftp полезности, для IPv4r2 ftp сервиса совершенно необходимо иметь такую реализацию ftp клиента и сервера, которая может работать в условиях IPv4r2, когда ftp сервера не знают своих IPv4 внешних адресов, для этого в IPv4r2 ftp в дополнение к IPv4 режиму PASSIVE вводится режим R2, так что участники для передачи данных без необходимости дополнительно не обмениваются между собой никакими адресами, а особенно IPv4 формата (см. описание ftp R2).

Как и IPv4 ftp, все IPv4 протоколы, которые работают с глобальной IPv4 адресацией или передают внутри себя глобальные IPv4 адреса, для связи по IPv4r2 нуждаются в специальных режимах работы, которые исключают встроенное использование IPv4 адресации.

Протоколы, в которые жестко встроена IPv4 адресация, как минимум несовершенны. В качестве адресов для обмена в хорошем протоколе могут использоваться только:

- глобальные URL, независимые от IP, которые заданы в виде имен, по которым любой клиент любой IP сети может получить доступ к этим ресурсам;
- или те адреса, которые получены из IPv4 заголовка.

Для протокола IPv4r2, все IPv4 протоколы которые отправляют внутри себя IPv4 адреса ресурсов, должны отправлять URL этих ресурсов имеющие смысл в глобальной сети, а клиенты отправлять серверам URL этих серверов, под которыми эти сервера были доступны со стороны клиента.

FTP режим передачи данных R2.

В IPv4r2 для IPv4 ftp вводится новый ftp режим передачи данных R2, так что участники ftp обмена при передаче данных дополнительно не обмениваются между собой IPv4 адресами, а используют те локальные (имеющие смысл только для их локальных машин) IPv4 адреса, с помощью которых была установлена IPv4 ftp связь в глобальной IPv4r2 сети. Это позволяет IPv4 ftp участникам обмениваться данными в глобальных IPv4r2 сетях, когда ни для одного из них не существует внешнего глобального IPv4 адреса.

Для режима R2 протокола ftp справедливо, что если вам удалось осуществить связь с ftp сервером авторизации, это означает вы сможете (т.е. исключены трудности связанные с IP адресацией) в пассивном или активном режиме обмениваться данными с этим ftp сервером, который также должен выполнять и функции ftp сервера данных, что и существует на практике для реальных ftp серверов в малых локальных сетях.

Перед каждой каждой командой PORT или PASSIVE для настройки передачи данных, ftp клиент выдает ftp серверу ftp команду "R2" и ожидает ftp ответ "327 R2, data transfer uses the same IP.", за которым идет обычный порядок IPv4 ftp команд. Для включения и отключения такого режима есть одноименная команда R2 для ком- строки minix ftp клиента.

В случае использования PASSIVE режима, в некоторых случаях передача данных по ftp будет возможна, когда поддержка режима R2 есть только у ftp клиента, который не будет посылать ftp серверу команду R2, но будет игнорировать IPv4 адрес, присылаемый IPv4 ftp сервером. Для включения такого режима команда R2 ком- строки minix ftp клиента вызывается с параметром 0: "R2 0".

Работа IPv4 TALK в сети "IPv4r2.0".

Работа minix реализации talk также связана с использованием глобальной IPv4 адресации для внутренних нужд, поскольку протокол talk не имеет такого важного значения как протокол ftp, то правок talk сервера и клиента для minix можно не делать, а использовать talk поверх telnet/rlogin/rsh соединений, которые не используют глобальную IPv4 адресацию для внутренних нужд и работают в сетях IPv4r2 без модификаций.

IPv4r2 адреса в IPv4 опциях маршрутизации и отладки.

Ряд опций IPv4 для маршрутизации и отладки предполагают хранение в IPv4 заголовке IPv4 адресов. Поскольку адреса IPv4r2 много- уровневые и обычно занимают 9 или 10 байт, то записи маршрутов и адресов в сети в виде IPv4r2 адресов это довольно нерациональная операция с учетом MTU 1152.

Если запись адресов такого рода (опциями IPv4) нужна, то все маршрутизаторы IPv4r2, которые отмечаются в IPv4r2 заголовке, должны иметь адреса или в корневой IPv4 сети, или в локальной IPv4 сети для каждого предыдущего IPv4r2 маршрутизатора (для фиксированной маршрутизации это условие выполнить легче всего), поэтому эти IPv4 опции не подвергаются в IPv4r2 модификации - они хранят локальные адреса в IPv4 сетях.

Из назначения "межсетевого протокола" следует, что корневая сеть IPv4 должна прямо адресовать только шлюзы корневой сети, а адресация машин или подсетей, скрытых за этими шлюзами, должна достигаться внутренними протоколами, которые способны сформировать многоуровневый в терминах IPv4 адрес.

В версии "межсетевого протокола" IPv4r2, этот многоуровневый IPv4r2 адрес машин и подсетей хранится в целом аналогично опциям маршрутизации IPv4 заголовка, так что протоколы верхнего уровня часто смогут могут работать в IPv4r2 сетях также как и в корневой сети IPv4.

Если нужна более сложная, глобальная маршрутизация или маршрутизация с указанием адресов IPv4r2, нужно использовать специальный протокол маршрутизации заместо IPv4 опций внутри IPv4r2 заголовка.

IPv4 опции 0x00, 0x01.

Эти одно- байтовые опции служат для выравнивания остальных опций IPv4 заголовка по границе 4 байт. Если выравнивание происходит в промежуточных блоках, то используется код 0x01, если же выравнивание происходит в последнем блоке, то используется код 0x01 или 0x00.

Код 0x00 завершает список опций, независимо от того, что идет далее. Может использоваться для оперативного ограничения списка опций на последовательных этапах обработки опций маршрутизатором, когда размер блока опций в заголовке еще не пересчитан. В готовых для отправки по сети IPv4r2 пакетах рекомендуется использовать последовательность кодов 0x00 (от 1 до 3) только для записи в конец блока опций для выравнивания блока опций по границе 4 байт.

Код 0x01 используется тогда, когда новые опции вставляются перед или между уже имеющимися и уже выровненными блоками опций так, чтобы не сканировать и не удалять уже имеющиеся опции на предмет поиска опций выравнивания, т.е. формируется не самый оптимальный пакет, который имеет несколько лишних байт 0x01, но зато при этом отправителю не нужно проводить полный просмотр опций и удаление имеющихся опций выравнивания, т.е. не надо делать полную пере- сборку опций.

Также код 0x01 используется для быстрого исключения опций малого размера в промежуточных маршрутизаторах, например, шлюз провайдера может не пере- собирать все опции для удаления трех- байтовой команды для шлюза, а после ее обработки "забить" эту опцию тремя байтами 0x01.

В протоколе IPv4r2 не допускается использование опций IPv4 с кодами 0x01 для задания опкода, т.е. так, что без выравнивания кодами 0x01? после некоторых опций следующие опции меняют смысл или не могут быть прочитаны. Выравнивание опций IPv4r2 происходит по границе байта, а всего блока опций по границе 4 байта.

Опции 0x01 и 0x00 вспомогательные и могут быть вставлены или удалены в любой момент по потребностям при продвижении пакета по IPv4r2 сети.

Эффективность использования IPv4r2 адресов в IPv4 опциях.

Для определения IPv4r2 адресов, маршрутизатор в сетях IPv4r2 должен проводить просмотр и анализ списка IPv4r2 опций, что в принципе дает значительное пенальти в виде потери производительности таких систем по сравнению с системами, где адрес указан в фиксированном поле.

Для решения этой проблемы и для улучшения внутренней структуры IPv4r2 заголовка, при использовании IPv4r2 опций расширения адреса, IPv4r2 адресаты по возможности помещают адресные IPv4r2 опции основной адресации (базовой или обобщенной) по фиксированному смещению 20 относительно начала заголовка, т.е. расположение больших адресов в этом формате IPv4r2 заголовка становится фиксированным, при этом сам заголовок остается полностью совместимым с IPv4.

Для каждого IPv4r2.x существует свой эффективный порядок адресных опций (адресный список).

При использовании основных схем кодирования IPv4r2 адресации, расширение адресов и индексы шлюза кодируются единственной опцией opX, что дополнительно удовлетворяет этим требованиям и если сравнивать такой IPv4r2 пакет с идеальным IPv4 пакетом без опций, то сканирование единственной опции не сильно отличается от фиксированной позиции в заголовке.

Заметим, что если опции присутствуют в IPv4r2 заголовке, то эти опции в общем все равно придется сканировать (ну, может быть не в момент определения IPv4r2 адресов), так что запись адреса в опции это не такая и большая утрата производительности на самом деле.

Также если информация, которая записана в опциях, существенна для маршрутизации, то по большому счету это все равно, записана ли эта информация в виде списка опций или по фиксированным смещениям, особенно если запись по фиксированным смещениям в принципе невозможна.

При трансляции IPv4r2 пакетов по сетям IPv4, порядок опций заголовка в промежуточных IPv4 маршрутизаторах может меняться, поэтому каждый IPv4r2 маршрутизатор, обнаружив неправильный порядок IPv4r2 опций, может восстановить эффективное расположение опций при трансляции IPv4r2 пакета в следующую подсеть, таким образом дополнительное падение производительности из-за перестановки опций будет только при трансляции IPv4r2 пакетов через "плохие" с точки зрения IPv4r2 сетей маршрутизаторы IPv4.

Алгоритм поиска расширения адреса в опциях может быть таким:

- просмотреть список опций входящего IPv4r2 пакета и в зависимости от целей маршрутизатора:
  - построить IPv4r2 псевдо- заголовок из важных полей (адресных например);
  - восстановить правильный порядок опций в ретранслируемом IPv4r2 пакете, если этот порядок был нарушен.

В каких-то случаях маршрутизатор, проверив во входящем IPv4r2 пакете опции, найдя правильный порядок опций и обнаружив по смещению 20 единственную базовую кодировку IPv4r2 (все остальные опции могут быть для этого маршрутизатора не обслуживаемыми), может отправить такой IPv4r2 пакет на обработку своему высокопроизводительному коду, который будет считывать базовую адресацию непосредственно из IPv4r2 заголовка. Этот случай в целом мало чем отличается от фиксированных смещений IPv6.

Опция дополнительной IPv4r2 адресации пользовательской сети должна быть задана второй, сразу после задания основной адресации (базовой или обобщенной). Опция IPv4r2 описания подсетей должна быть задана третьей, сразу после адресации пользовательской сети. Эти опции могут не использоваться промежуточными IPv4r2 маршрутизаторами, их анализируют только конечные IPv4r2 адресаты.

При восстановлении порядка IPv4r2 опций в промежуточных маршрутизаторах, дополнительные IPv4r2 опции, анализ значений которых требует сложной интерпретации их битовых полей, могут просто помещаться после опций основной адресации, декодирование которых простое, в начало IPv4r2 заголовка.

Полное описание расширения IPv4 до IPv4r2, тонкая настройка IPv4r2.

Адресация IPv4r2.

-) Многоуровневый режим IPv4r2 адресации.

По сравнению с 128 битным адресом IPv6, 128 битный адрес IPv4r2 имеет четырех- уровневую IPv4 иерархию:

```
a1.b1.c1.d1/a2.b2.c2.d2/a3.b3.c3.d3/a4.b4.c4.d4
```

Все базовые режимы основной IPv4r2 адресации (77 бит, 80 бит и 72 бита) отображаются на 128 битный адрес четырех-уровневой иерархии IPv4 адресов по десяти- байтовой схеме 4/2/3/1 так:

```
a1.b1.c1.d1/c2.d2/b3.c3.d3/d4
```

специальным образом заполняя эти байты. Часть битов d4 этой упрощенной схемы на деле попадают в a3, т.е. d4 и a3 не используют все свои 8 бит (см. описание основной IPv4r2 адресации далее).

А обобщенное IPv4r2 основное расширение адресации отображается на 128 битный адрес четырех- уровневой иерархии IPv4 адресов иначе:

- биты поля расширения адреса от старшего бита к младшему биту образуют байты;
- байты располагаются в 128 битном IPv4r2 адресе по уровням, полностью заполняют один уровень, без пропусков, только потом переходят к следующему уровню;
- первыми в поле расширения адреса идут байты старших уровней, старшинство убывает с возрастанием номеров уровня (второй, третий, четвертый);
- в пределах одного уровня байты идут от младшего к старшему: dX, cX, bX, aX;
- для асимметричной адресации 70 это выглядит так
  a1.b1.c1.d1/a2.b2.c2.d2/a3.b3.c3.d3/d4
  d4[6] = поле <младший байт>[6],
  <d2>[8]<c2>[8]<b2>[8]<a2>[8]<d3>[8]<c3>[8]<b3>[8]<a3>[8] = поле <адрес>[64]
- для асимметричной адресации 38 это выглядит так
  a1.b1.c1.d1/a2.b2.c2.d2/d3
  d3[6] = поле <младший байт>[6],
  <d2>[8]<c2>[8]<b2>[8]<a2>[8] = поле <адрес>[32]

В асимметричном режиме обобщенном IPv4r2 расширении адресации, поле младший байт адреса (d2 для 6 бит), если не равно нулю, то выравнивается по границе младшего бита, заполняя старшие биты неполно адресуемого младшего байта нулями. Если поле младший байт адреса равно нулю, то оно считается не используемым и не входит в число байт длины адреса (см. ниже сравнение адресов).

-) Сравнение IPv4r2 адресов.

Адреса IPv4r2 в базовом и обобщенном режимах IPv4r2 основной адресации принадлежат одному и тому же основному IPv4r2 адресному пространству. Адреса в этом основном IPv4r2 адресном пространстве имеют длину, выражаемую в целых байтах. При сравнении таких адресов равными могут быть только адреса с равной длиной, независимо от того, как совпадают их битовые поля на участках общей длины для адресов разного размера. Это дополнительно увеличивает адресное пространство IPv4r2, которое независимо для IPv4r2 адреса каждой длины.

Адреса IPv4r2 в режиме дополнительной IPv4r2 адресации сети пользователя образуют отдельное от основного IPv4r2 адресное пространство, которое также имеет длину адреса. Сравнение дополнительного IPv4r2 адреса производится по тем же принципам как и основного, после того, как совпал основной адрес.

Адреса IPv6 в режиме дополнительной IPv6 адресации в сети IPv4r2 также образуют свое отдельное от основного IPv4r2

адресное пространство, которое также имеет длину адреса фиксированного размера 128 бит. Сравнение дополнительного IPv6 адреса производится по правилам IPv6, после того, как совпал основной адрес.

Адреса в базовых режимах основной IPv4r2 адресации имеют реальную длину 9 или 10 байт, но отображаются на 16 байтовое (128 битное) четырех- уровневое представление, а в режиме обобщенного основного расширения адресации, адреса непрерывно отображаются до того уровня, до которого адрес заполняется байтами адреса, дополняя старшие байты IPv4 адреса последнего уровня нулями (в примере 0.0.0.d2). Другими словами, гранулярность (минимальное приращение) длины обобщенного основного адреса равна четырем байтам, независимо от того сколько байт хранится в поле адреса заголовка. При упаковке обобщенного основного адреса в IPv4r2 заголовок, старшие байты последнего уровня могут быть отброшены, если равны нулю (три байта в этом примере).

Поэтому чтобы выразить базовые адреса с помощью обобщенной адресации, размер обобщенного адреса должен быть равен 13 байтам (102 бит), а не 9 или 10 байт (48 бит) как для базовой адресации. Для базовой основной IPv4r2 адресации нет проблемы вычисления и сравнения длины адреса, это всегда адрес 16 байт (128 бит) длиной в виде 9 или 10 реально используемых байт.

Таким образом, обобщенное расширение адресации позволяет обращаться в пределах 128 бит к глобальным адресам, которые недостижимы с помощью базовых режимов адресации, но практического смысла в такой адресации нет, поскольку базовая адресация итак уже очень велика.

-) Режим IPv4r2 адресации 77 бит (главный режим IPv4r2 нормальной базовой адресации, используется при асимметричной адресации).

Этот 77 битный адрес отображается на 128 битный адрес четырехуровневой иерархии IPv4 адресов:

a1.b1.c1.d1/a2.b2.c2.d2/a3.b3.c3.d3/a4.b4.c4.d4
как
a1.b1.c1.d1/c2.d2/a3.b3.c3.d3/d4

Биты поля <расширения адреса>[45] от старшего бита к младшему биту располагаются по группам:
<d4>[3]<a3.b3>[10]<c3.d3>[16]<c2.d2>[16]

Сеть d4 из 8 адресов рассматривается как:

- 3 битная IPv4r2 подсеть с хостами 0-7 (обратная маска /:-3, 0,7 разрешены как адреса хоста)

Не могут образовать IPv4 подсеть.

а) корневая IPv4 сеть (первый уровень)

в которой IPv4 адрес a1.b1.c1.d1, 32 бита
выдаются так же, как и сейчас в интернете
эти a1.b1.c1.d1 адреса являются точками входа в деревья IPv4r2 сетей
эти деревья бывают разных типов по своей структуре, например:
a1.b1.c1.1 это региональная IPv4r2 структура (классификация по странам, подобно ru/ua доменным корням)
a1.b1.c1.2 это функциональная IPv4r2 структура (классификация по типу, подобно com/org доменным корням)
и т.д.

Наличие корневой IPv4 сети для адресации IPv4r2 позволяет обычным сетям IPv4 по крайней мере доставлять IPv4r2 пакеты в адрес IPv4 точек входа в сети IPv4r2 адресации, а шлюзы IPv4, которые маршрутизируют пакеты на пути до этих IPv4 точек входа и сами эти IPv4 точки входа уже должны быть подсоединены к сетям IPv4r2 и уметь доставлять пакеты анализируя полный IPv4r2 адрес.

Далее рассмотрим IPv4r2 сеть с региональной структурой.

б) второй уровень
в которой IPv4 адрес c2.d2, 16 бит
формат c2.d2 зависит от структуры этой IPv4r2 сети (от корневой точки входа a1.b1.c1.d1)
для IPv4r2 сети региональной структуры
c2.d2 имеют формат <региональный код>[10]<региональная сеть>[6]
так что что для каждой точки входа a1.b1.c1.d1 предоставляется
1024 региона Земли и 64 региональные сети в каждом таком регионе (64 региональных провайдера)
адреса c2.d2 выдаются владельцем корневой точки входа a1.b1.c1.d1

в) третий уровень
в которой IPv4 адрес a3.b3.c3.d3, 26 бит
в сети IPv4r2 региональной структуры
для каждого регионального провайдера c2.d2
адреса a3.b3.c3.d3 имеют формат <клиент>[26]
это 64 миллиона клиентов регионального провайдера
адреса <клиент> выделяются региональным провайдером c2.d2 своему клиенту

г) четвертый уровень
в которой IPv4 адрес d4, 3 бита
в сети IPv4r2 региональной структуры
адрес d4 имеет формат <сеть пользователя>[3]
для каждой a1.b1.c1.d1 IPv4r2 сети региональной структуры

для каждого регионального провайдера c2.d2,
для каждого клиента регионального провайдера a3.b3.c3.d3,
<сеть пользователя> это d4 сеть клиента с 8 адресами
адреса <сеть пользователя> распределяет сам пользователь на свои ресурсы

Таким образом, полный вид IPv4r2 асимметричного адреса назначения в IPv4r2 сети региональной структуры такой:
a1.b1.c1.d1[32]/c2.d2[16]/a3.b3.c3.d3[26]/d4[3] [всего 77 бит]
даже если занять только несколько адресов a1.b1.c1.d1 в корневой сети IPv4 под точки входа в такие IPv4r2 сети
это уже даст сотни миллиардов глобальных IPv4r2 адресов с асимметричной адресацией
которых хватит на всю солнечную систему даже без модификации IPv4 заголовка

-) Режим IPv4r2 адресации 80 бит (режим IPv4r2 длинной базовой адресации).

Биты поля <расширения адреса>[48] от старшего бита к младшему биту располагаются по группам:
<d4>[6]<a3.b3>[10]<c3.d3>[16]<c2.d2>[16]

Сеть d4 из 64 адресов рассматривается как:

- 6 битная IPv4r2 подсеть с хостами 0-63 (обратная маска /:-6, 0,63 разрешены как адреса хоста)

Первые 8 хостов из 64 те же что в режиме 77 бит. Не могут образовать IPv4 подсеть.

В сетях IPv4r2 региональной структуры этот режим отличается от предыдущего 77 битного только тем, что
на четвертом уровне 6 бит вместо 3,
т.е.:
для каждой a1.b1.c1.d1 IPv4r2 сети региональной структуры
для каждого регионального провайдера c2.d2
есть 64 миллиона a3.b3.c3.d3 клиентов
с d4 сетями по 64 адреса

Таким образом, полный вид IPv4r2 адреса назначения в IPv4r2 сети региональной структуры такой:
a1.b1.c1.d1[32]/c2.d2[16]/a3.b3.c3.d3[26]/d4[6] [всего 80 бит]
даже если занять только несколько адресов a1.b1.c1.d1 в корневой сети IPv4 под точки входа в такие IPv4r2 сети
это уже даст сотни миллиардов глобальных IPv4r2 адресов с адресацией 80 бит
которых хватит на всю солнечную систему даже без модификации IPv4 заголовка

-) Режим IPv4r2 адресации 72 бит (режим IPv4r2 короткой базовой адресации).

Биты поля <расширения адреса>[40] от старшего бита к младшему биту располагаются по группам:

<d4>[2]<b3>[6]<c3.d3>[16]<c2.d2>[16]

Сеть d4 из 4 адресов рассматривается как
- 2 битная IPv4r2 подсеть с хостами 0-3 (обратная маска /:-2, 0,3 разрешены как адреса хоста)

Это первые 4 из 8 тех же хостов что и в режиме 77 бит. Не могут образовать IPv4 подсеть.

В сетях IPv4r2 региональной структуры этот режим отличается от предыдущего 77 битного только тем, что
на третьем уровне 22 бита вместо 26,
а на четвертом уровне 2 бита вместо 3,
т.е.:
для каждой a1.b1.c1.d1 IPv4r2 сети региональной структуры
для каждого регионального провайдера c2.d2
есть 4 миллиона b3.c3.d3 клиентов
с d4 сетями по 4 адреса

Таким образом, полный вид IPv4r2 адреса назначения в IPv4r2 сети региональной структуры такой:
a1.b1.c1.d1[32]/c2.d2[16]/b3.c3.d3[22]/d4[2] [всего 72 бит]
даже если занять только несколько адресов a1.b1.c1.d1 в корневой сети IPv4 под точки входа в такие IPv4r2 сети
это уже даст сотни миллиардов глобальных IPv4r2 адресов с адресацией 72 бита
которых хватит на всю солнечную систему даже без модификации IPv4 заголовка

-) Обобщенное IPv4r2 расширение адресации.

В обобщенном расширении IPv4r2 основной адресации, адреса внутренних уровней (это имеет значение для коротких адресов) раздает владелец корневого адреса (a1.b1.c1.d1).

Адреса, численно совпадающие с базовой IPv4r2 основной адресацией, хотя и достижимы в кодировании обобщенной IPv4r2 основной адресации, но распределяются в рамках базовой IPv4r2 основной адресации.

-) Дополнительная IPv4r2 адресация пользовательской сети.

Повторим, что это отдельное адресное пространство, распределение адресов в котором осуществляет сам владелец выделенного IPv4r2 адреса для своих ресурсов.

При маршрутизации пакеты в адрес дополнительной сети пользователя передаются на основной IPv4r2 адресат, для которого этот адрес дополнительной сети определен, но в отличие от индекса локального сокета шлюза, расширять можно оба адресата одновременно.

При работе через через шлюз провайдера локальный адрес дополнительной сети пользователя может также подменяться, как и локальный IPv4/IPv4r2 адрес.

Работа IPv4r2 в полном режиме.

-) Подсчет контрольной суммы IPv4r2 заголовка.

Чтобы указать, что подсчет IPv4 контрольной суммы в этом заголовке IPv4r2 не производился, создатель IPv4r2 пакета записывает значение 0xFFFF в поле контрольной суммы этого IPv4r2 заголовка.

Причины возможности резервирования значения 0xFFFF в поле контрольной суммы IPv4r2 заголовка:

- суммирование IPv4 заголовка производится так, что в результате сумма может быть равна 0 только если все поля IPv4r2 заголовка равны 0, а поскольку как минимум поле версии заголовка содержит число 4, то сумма 0 никогда не может получиться;
  - в поле контрольной суммы IPv4r2 заголовка записывается инверсия этой суммы, так чтобы при сложении получилось значение 0xFFFF, значит раз сумма не равна 0x0000, то значение 0xFFFF никогда не может появляться в поле контрольной сумме IPv4r2 заголовка и может использоваться как флаг.

Проблемы резервирования значения 0xFFFF:

- отказ IPv4r2 от вычисления контрольной суммы не является 100% совместимым с IPv4, который предполагает ее вычисление, если контрольная сумма рассматривается IPv4 маршрутизатором как поврежденная (на деле в ней записано резервное IPv4r2 значение 0xFFFF), то такой IPv4 хост не сможет отвечать ICMP ответом, поскольку отправитель неизвестен, либо IPv4 должен уметь отвечать ICMP ответом на стандартный широковещательный адрес ошибок (предыдущий шлюз находится в той же сети, где и IPv4 маршрутизатор), отправляя туда IPv4r2 заголовок;
- это нерационально расходует 2 байта IPv4r2 заголовка, позволяя этим только не считать контрольную сумму;

Старое оборудование, переносящее IPv4 пакеты, становится разделенным на два класса: совместимое с IPv4r2 по интерпретации поля контрольной суммы и несовместимое. Как бороться с несовместимостью?

a) Маршрутизатор IPv4r2 должен знать, поддерживает ли промежуточный IPv4 адрес, куда он отправляет пакет для IPv4r2 адресата, IPv4r2 отказ от подсчета контрольной суммы IPv4 заголовка.

b) Модификация модификации рознь. Для исправления программ, обслуживающих оборудование поддерживающее исходный IPv4, с целью внесения совместимости по IPv4r2 интерпретации поля контрольной суммы, изменения будут откровенно косметическими (по сравнению с ситуацией внедрения IPv6), даже "решение в лоб" путем:
   - фильтрации всех входящих IPv4 пакетов с полем контрольной суммы 0xFFFF на предмет подсчета контрольной суммы;
   - фильтрации всех исходящих IPv4 пакетов установкой поля контрольной суммы в значение 0xFFFF;

без учета эффективности, равной занесению правильных значений в момент создания IPv4r2 заголовка.

Так вот, изменения для программ с открытым исходным кодом, связанные с использованием резервного значения 0xFFFF в поле контрольной суммы, будут очень простыми, для модификации не потребуется смена ядра, переделка всего стека протоколов и не нужны т.п. глобальные обновления.

-) Формат IPv4r2 заголовка в основном режиме.

В полном режиме IPv4r2 заголовок может задействовать резервный бит флагов IPv4 заголовка, устанавливая этот бит в 1.

Установка в 1 резервного бита флагов IPv4 заголовка указывает на формат IPv4r2 заголовка. Это приводит к следующему:

- в IPv4 поле протокол резервируется специальное значение номера протокола=0 (двоичное число 0)
  - если номер протокола не равен 0, то
    - предполагается работа IPv4r2 протокола в режиме межсетевого взаимодействия с MTU 1152 (работа в интернет);
    - поле протокола имеет такое же значение, как и для IPv4 заголовка
  - если номер протокола равен 0, то
    - предполагается работа IPv4r2 в режиме локальных и специальных сетей с MTU больше чем 1152
    - используется формат расширения статических полей IPv4r2 заголовка (см далее обобщенное расширение IPv4r2 заголовка);

Установка в 1 резервного бита флагов и поле протокол не равно 0 (MTU 1152):

- запрещает IPv4 фрагментацию;
- запрещает подсчет IPv4 контрольной суммы;
- включает асимметричный 45 бит режим базовой IPv4r2 адресации (основной IPv4r2 режим межсетевого взаимодействия 77 бит), при этом освободившиеся поля IPv4 заголовка передаются под IPv4r2 поле расширение адреса (дополнительно 45 бит), состоящее из:
  - IPv4 поле контрольной суммы
    хранит <c2.d2>[16]
  - IPv4 поле идентификатора
    хранит <c3.d3>[16]
  - IPv4 поле смещения сегмента
    хранит <d4>[3]<a3.b3>[10]
- IPv4 поле флаги имеет новое значение:
  - маска 100b (IPv4 резервный флаг)
    - 1

признак IPv4r2 заголовка
флаги D,M имеют специальное, описанное здесь значение
- маска 010b (IPv4 D флаг)
  - 0
    поле расширение адреса 45 бит IPv4r2 заголовка принадлежит адресу назначения
  - 1
    поле расширение адреса 45 бит IPv4r2 заголовка принадлежит адресу источника вместо адреса назначения
- маска 001b (IPv4 M флаг)
  - 0
    максимальный размер IPv4r2 заголовка до 60 байт
  - 1
    расширение максимального размера IPv4r2 заголовка до 80 байт

Асимметричный 45 бит режим базовой IPv4r2 адресации предназначен для работы с IPv4r2 серверами пользователя в сети интернет через шлюз провайдера, поэтому по умолчанию расширяется поле адреса назначения. При отправке обратных пакетов включается D флаг (маска 010b), который указывает что расширяется поле источника.

Фактически такой формат IPv4r2 заголовка позволяет использовать IPv4r2 глобальную адресацию не используя опции IPv4 заголовка, т.е. при использовании 77 битной IPv4r2 адресации IPv4r2 заголовки имеют размер 20 байт, как и для 32 битных IPv4 адресов. Если сеть провайдера поддерживает IPv4r2, то этот формат IPv4r2 заголовка является наиболее часто используемым.

Расширение размера IPv4r2 заголовка до 80 байт прибавляет число 20 к размеру заголовка хранящемуся в IPv4r2 поле размер заголовка, т.е.:
- IPv4r2 поле размер заголовка=5..F означает IPv4r2 заголовок=5*4..15*4=20..60 байт;
- IPv4r2 поле размер заголовка=0..F в режиме расширения до 80 байт означает IPv4r2 заголовок=20+(0*4..15*4)=20..80 байт.

Проблемы в использовании резервного бита флагов:
- отметим, что существует rfc3095, предлагающее использовать этот резервный бит флагов для "передачи IPv4 пакетов в сетях сотовой связи", но доставка пакетов в сетях "точка-точка" позволяет обертывать IP пакеты в дополнительную транспортную оболочку, которая никак не связана с IP заголовком и может позволять как сжимать IP пакеты в целом, так и добавлять средства контроля целостности данных;
- также отметим, что сотовые компании не очень заинтересованы в том, чтобы трафик с абонента был бы минимальным, что для "общества свободного рынка не уравновешенного системой ценностей" есть ситуация обычная, если один абонент займет весь канал, владелец сети получит столько же, как если бы канал был занят тысячей абонентов - для узких

каналов продается их пропускная способность, а не безлимитный доступ (количество абонентов);

- нам выгодно, включив резервный флаг IPv4, добиться изменения формата IPv4 заголовка для нужд IPv4r2 и использовать освободившиеся IPv4 поля для расширения IPv4r2 адресации.

Старое оборудование, переносящее IPv4 пакеты, становится разделенным на два класса: совместимое с IPv4r2 по использованию IPv4 резервного флага и протоколом не равным 0 и несовместимое. Как бороться с несовместимостью?

a) Маршрутизатор IPv4r2 должен знать поддерживает ли промежуточный IPv4 адрес, куда он отправляет пакет для IPv4r2 адресата, использование IPv4 резервного флага и протокола не равного 0 или нет.

b) Правильное поведение шлюза IPv4, когда он встречает IPv4r2 пакет с резервным битом в неправильном положении, это отбросить такой IPv4 пакет, поскольку формат IPv4 заголовка для такого взведенного бита, кроме поля версия и резервного бита поля опции, может быть произвольным (в описании IPv4 формат заголовка с таким битом не определен). Но существуют IPv4 реализации (или опции настройки такого поведения), которые полагают, что резервные биты можно безопасно игнорировать, а не отбрасывать пакеты с такими битами в неправильном состоянии.

c) Модификация модификации рознь. Для исправления программ, обслуживающих оборудование поддерживающее исходный IPv4, с целью внесения совместимости по IPv4r2 интерпретации IPv4 резервного флага поля флагов и протокола не равного 0, изменения будут откровенно косметическими (по сравнению с ситуацией внедрения IPv6), даже "решение в лоб" путем:
   - фильтрации всех входящих IPv4 пакетов на предмет проверки IPv4 установленности резервного флага и протокола на неравенство 0 и переноса адресных полей из IPv4 заголовка в IPv4r2 опции;
   - фильтрации всех исходящих IPv4 пакетов на предмет установки IPv4 резервного флага и переноса адресных полей из IPv4r2 опций в IPv4 заголовок;
   - расширение размера IPv4r2 заголовка до 80 байт во многих случаях не потребует никакой существенной переделки, кроме добавления исправленной копии оригинальной IPv4 функции, обрабатывающей максимальный размер IPv4 заголовка и механизма выбора функции в зависимости от флага заголовка.

-) Обобщенное расширение IPv4r2 заголовка.

Обобщенное расширение размера IPv4r2 заголовка предназначено:

- для работы IPv4r2 в локальных и специальных сетях, когда MTU больше чем 1152.

Как и в случае расширение максимального размера IPv4r2 заголовка до 80 байт, это обобщенное расширение не предназначено для замены специальных протоколов опциями IPv4.

Для обобщенного расширения максимального размера IPv4r2 заголовка:

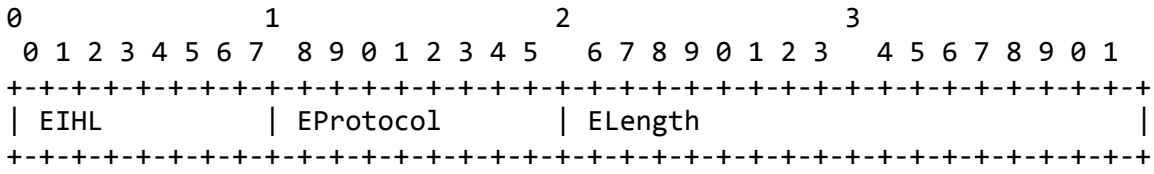
- должен быть включен резервный бит IPv4 поля флагов;
- в IPv4 поле протокол записывается номер специального протокола=0;

важно отметить, что при этих условиях заголовок IPv4r2 не состоит из двух раздельных заголовков двух разных протоколов,

а состоит только из одного заголовка одного протокола IPv4r2, значение 0 в IPv4 поле протокола указывает что для IPv4r2 это вовсе не поле протокола.

Установка в 1 резервного бита флагов и поле протокол равно 0 (MTU более 1152):

- запрещает IPv4 фрагментацию;
- запрещает подсчет IPv4 контрольной суммы;
- освободившиеся поля IPv4 заголовка зарезервированы:
  - IPv4 поле контрольной суммы
    зарезервировано <0>[16]
  - IPv4 поле идентификатора
    зарезервировано <0>[16]
  - IPv4 поле смещения сегмента
    зарезервировано <0>[13]
- IPv4 поле флаги имеет новое значение:
  - маска 100b (IPv4 резервный флаг)
    признак IPv4r2 заголовка
  - маска 010b (IPv4 D флаг)
    резерв 0
  - маска 001b (IPv4 M флаг),
    резерв 0
- к фиксированной части IPv4r2 заголовка прибавляются новые поля, идущие заместо IPv4 списка опций, начиная со смещения 20:



  - поле EIHL
    содержит значение размера дополнительного заголовка в 4 байтовых словах
    размер EIHL включает в себя и сами новые поля (минимальное значение EIHL = 1)
    значение IHL поля размера IPv4 заголовка при этом расширении всегда = 5
    максимальный размер IPv4r2 заголовка равен 1020(EIHL)+20(IHL)=1040 байт
  - поле EProtocol
    содержит протокол который должен быть в поле IPv4 заголовка Protocol, если не применяется обобщенное расширение IPv4r2 заголовка

значение EProtocol=0 зарезервировано (также как для поля Protocol заголовка IPv4)
    - если EProtocol не равен 0, то после новых статических полей IPv4r2 заголовка идет поле списка опций, аналогичное IPv4 заголовку;
    - если EProtocol равен 0, то формат IPv4r2 пакета не определен;
  - поле ELength
    старшие два байта размера IPv4r2 пакета (младшие два байта в поле размер пакета IPv4 заголовка)
    максимальный размер IPv4r2 пакета при этом 4Гбайта

В режиме локальной и специальной сети асимметричная IPv4r2 адресация по умолчанию не задействована, IPv4r2 адресация достигается с помощью IPv4r2 адресных опций записанных в IPv4r2 поле опций (после новых статических полей IPv4r2 заголовка). Для локальной IPv4 сети будет достаточно IPv4 адресов IPv4 локальной сети (являющихся адресами корневой IPv4r2 сети).

Старое оборудование, переносящее IPv4 пакеты, становится разделенным на два класса: совместимое с IPv4r2 по использованию IPv4 резервного флага и протокола равного 0 и несовместимое. Как бороться с несовместимостью?

a) Маршрутизатор IPv4r2 должен знать поддерживает ли промежуточный IPv4 адрес, куда он отправляет пакет для IPv4r2 адресата, использование IPv4 резервного флага или нет.
b) Модификация модификации рознь. Для исправления программ, обслуживающих оборудование поддерживающее исходный IPv4, с целью внесения совместимости по IPv4r2 интерпретации IPv4 резервного флага поля флагов и протокола равного 0, изменения будут откровенно косметическими (по сравнению с ситуацией внедрения IPv6):
    - поддержка IPv4r2 пакетов 4Гбайта во многих случаях не потребует никакой существенной переделки, кроме добавления исправленной копии оригинальной IPv4 функции, обрабатывающей максимальный размер IPv4 пакета и механизма выбора функции в зависимости от флагов заголовка.

Формат IPv4r2 URL.

URL для IPv4r2 имеет в общем такой вид:

- [протокол://] IPv4r2 адрес [:[порт] [:[индекс] [:[пользовательский адрес] [:[IPv6 адрес]]]]]

поля взятые в [] опциональны.

Форматы полей URL:

- IPv4r2 адрес:
  - IPv4[/IPv4[/...]]
    если в промежуточных уровнях есть IPv4 адреса a.b.c.d с нулевыми старшими байтами
    то они могут сокращенно записываться как "d/", "c.d/", "b.c.d/"
    например: IPv4/0/0/IPv4

таких IPv4 уровней для базовых адресов ровно 4, для обобщенных адресов 1 или более

- порт:
  - unsigned (произвольное число бит)
    порт HLAP или порт TCP/UDP
- индекс:
  - unsigned (произвольное число бит)
    индекс шлюза используется только IPv4r2 сервером
    в обратном обращении к клиенту (к источнику сеанса обмена)
- пользовательский адрес:
  - IPv4[/IPv4[/...]]
- IPv6 адрес:
  - IPv4r2 128 бит фиксированный формат IPv4/IPv4/IPv4/IPv4

Версии UDPr2/TCPr2 в стеке протоколов IPv4r2.

Альфа (предварительная) версия протоколов UDPr2/TCPr2, приведена для иллюстрации направлений развития IPv4r2 для этих протоколов.

-) Протокол локальной адресации хоста HLAP.

В стеке протоколов IPv4r2, вводится протокол локальной адресации хоста HLAP (аналогичен по функциям портам UDP/TCP).

- HLAP предположительно номер 0xA0

Заголовок HLAP протокола:

- поле локальный адрес(порт) источника 32 бит
- поле локальный адрес(порт) назначения 32 бит
- поле отметка времени 32 бит
- поле размер пакета 32 бит
- поле HLAP протокол 8 бит
- поле резерв для следующего протокола 24 бит

Размер заголовка фиксирован и составляет 20 байт.

Поле HLAP протокол содержит номера протоколов в формате HLAP, а не в формате IP (т.е. поверх HLAP нельзя отправить обычный IPv4 UDP/TCP сегмент).

Все поля заголовка HLAP неразрывны со следующим протоколом, указанным в поле HLAP протокол и устанавливаются этим

протоколом (т.е HLAP заголовок не может быть отправлен не инкапсулируя ничего):

- поле размер пакета содержит размер HLAP пакета (размер пакета следующего протокола плюс размер HLAP заголовка);
- если отметка времени не получена от следующего протокола (это поле равно 0), то устанавливается то время, когда отправляется HLAP заголовок;
- в поле резерв для следующего протокола данные размещает протокол следующего уровня (который указан в поле HLAP протокол).

В стеке протоколов IPv4r2, протоколы UDP/TCP имеют новые версии

- TCPr2 предположительно номер 0x06
- UDPr2 предположительно номер 0x11

опирающиеся на адресацию HLAP.

-) Контрольная сумма псевдо- заголовка.

Протоколы на базе HLAP:

- или совсем не подсчитывают контрольную сумму IPv4;
- или если подсчитывают, то не включают в псевдо- заголовок части IP заголовка, а используют только HLAP заголовок и блок данных самого пакета (попытка UDP/TCP протокола IPv4 включать IP сетевой адрес в псевдо- заголовок нарушает инкапсуляцию данных по уровням, HLAP полностью доверяет сетевую адресацию сетевому уровню, например IPv4r2).

-) Адресация портов UDPr2/TCPr2 увеличена с 16 до 32 бит.

Это нужно для поддержки шлюзов и защитных сетевых фильтров, делаем это потому, что все равно надо модифицировать заголовок, чтобы поддержать UDP/TCP размер пакета для локальных сетей до 4 Гбайт.

-) Протокол TCPr2.

Заголовок TCPr2 протокола:

- поле HLAP "резерв следующего протокола"
  - <размер опций заголовка>[3]<резерв 0>[15]<флаги>[6] =[24]
- порядковый номер 32 бит
- номер подтверждения 32 бит
- размер окна 32 бит
- указатель важности 32 бит
- поле опций

Размер фиксированной части TCPr2 заголовка составляет 16 байт (не включая часть от HLAP).

Поле размер опций заголовка принимая значения от 0 до 7 позволяет кодировать размер TCPr2 заголовка от 16 до 44 байт, что вместе с 20 байтами HLAP заголовка составляет 64 байта максимум, что подходит для MTU 1152 протокола IPv4r2. Если IPv4r2 заголовок при этом работает в режиме 80 байт, то максимальный размер для TCPr2 опций 8 байт (в поле размер опций заголовка значение 2), а полный размер заголовка TCPr2 + HLAP заголовка достигает предельных 48 байт.

-) Протокол TCPr2 теперь имеет поле отметка времени 32 бит.

Отметку времени проставляет источник TCPr2 сегмента, во время осуществления связи приемник может синхронизировать время источника и оценивать время отправления каждого пришедшего с момента установления связи сегмента.

-) Протокол TCPr2 теперь имеет поле размер пакета.

Поле размер пакета определяет размер пакета адресуемый TCPr2 полем порядковый номер, это потому что UDPr2/TCPr2 не требуют для своей работы какого-либо сетевого протокола (например, IPv4r2).

Обращаясь к локальному серверному порту, UDPr2/TCPr2 могут обращаться к серверу пользователя, который установил канал связи с машиной в локальной сети, т.е. может существовать иной, кроме IPv4r2, программный механизм отображения сетевых компьютеров на локальные порты, так что протоколам UDPr2/TCPr2 нет нужды знать о сетевых адресах и конфигурации сети, если пользователь нужные локальные сетевые ресурсы к локальным портам UDPr2/TCPr2 подключает сам.

Также сетевой протокол IPv4r2 может содержать служебные данные в IPv4r2 пакете, часть из которых не относится к данным UDPr2/TCPr2 или заголовку IPv4r2.

Т.е. для UDPr2/TCPr2 существует единственный уникальный сетевой адрес "локальная машина", которому не нужно присваивать фиктивный сетевой IPv4r2 адрес, это случается, когда сетевой адрес просто не указан.

-) TCPr2 поле размер пакета имеет размер 32 бит.

Это нужно для поддержки локальных сетей, размер пакета в байтах, размер TCPr2 пакета вырос до 4 Гбайт.

TCP опция Window scaling запрещена в TCPr2.

-) Поле TCPr2 "размер окна" увеличено с 16 до 32 бит.

Это нужно для поддержки локальных сетей, размер в байтах, размер TCPr2 пакета вырос до 4 Гбайт.

-) Поле TCPr2 "указатель важности" увеличено с 16 до 32 бит.

Это нужно для поддержки локальных сетей, размер в байтах, размер TCPr2 пакета вырос до 4 Гбайт.

-) Протокол UDPr2.

Заголовок UDPr2 протокола:

- поле HLAP резерв следующего протокола
  - <резерв 0>[8]<метка UDPr2 потока>[16] =[24]

Размер UDPr2 заголовка фиксирован и составляет 0 байт (не включая часть от HLAP).

Вместе с 20 байтами HLAP заголовка это 20 байт максимум, совместимо с режимом IPv4r2 заголовка 80 байт и с MTU 1152 протокола IPv4r2 без внимания о границе 128 байт.

-) Протокол UDPr2 теперь имеет поле отметка времени 32 бит.

Отметку времени проставляет источник UDPr2 дейтаграммы, приложение принимающее дейтаграммы может синхронизировать время источника и оценивать время отправления каждой прибывшей дейтаграммы.

-) UDPr2 поле размер пакета увеличено с 16 до 32 бит.

Это нужно для поддержки локальных сетей, размер в байтах, размер дейтаграммы увеличен до 4 Гбайт.

Расширения UDP/TCP в стеке протоколов IPv4r2.

-) Контрольная сумма псевдо- заголовка.

Протоколы UDP/TCP могут не подсчитывать контрольную сумму, помещая в поле контрольной суммы 0xFFFF, это возможно для UDP/TCP, поскольку поле длина пакета или смещение данных как минимум включает ненулевой размер пакета UDP/ заголовка TCP. Также протокол UDP может помещать в этом случае в поле контрольной суммы значение 0.

Далее, кроме отличий в заголовках, под "UDP/TCP" подразумеваются оба варианта протоколов UDP/TCP в сетях IPv4r2:

- старый UDP/TCP (0x11/0x06)
- новый UDPr2/TCPr2 (через HLAP)

-) TCP поле Window Size.

Протокол TCP имеет в поле Window Size значение

- 512 для IPv4
- 1024 для IPv4r2
- любое иное для локальных сетей
- 0 при перегрузке

По истечении таймера перегрузки, если подтверждения не было получено отправляется пробный пакет с размером:
- 512 для IPv4
- 1024 для IPv4r2
- любое иное для локальных сетей

TCP опция Maximum segment size имеет такие же правила 512/1024 для IPv4/IPv4r2 и поддерживает второй, 32 битный формат для локальных сетей:
- 2,6,SSSS

-) Арифметика TCP полей порядковый номер и номер подтверждения.

При переполнении эти поля подчиняются арифметике по модулю 2, а не той арифметике, что для контрольной суммы TCP.

-) Максимальный размер дейтаграммы UDP для IPv4r2.

UDP работающий локально в стеке IPv4r2 гарантирует, что как минимум 1024 байта самих данных в дейтаграмме может быть всегда принято самим протоколом UDP.

Для доставки UDP дейтаграмм через сеть IPv4r2 в межсетевом режиме IPv4r2, максимум 1024 байта данных в дейтаграмме может быть доставлено, если используются стандартные IPv4r2 (без расширения IPv4r2 заголовка или с расширением IPv4r2 заголовка до 80 байт) и UDP заголовки, иначе пакет не пройдет через IPv4r2 без фрагментации и может быть не доставлен до UDP назначения по причине IPv4r2.

Исправление IPv4r2 форматов опций в новых редакциях протокола.

В общем случае недопустимо менять предыдущие и анонсированные форматы кодирования, если только в них нет явной ошибки или изменение формата не повлияет на уже работающий софт. В этом подразделе приведен перечень несовместимых на двоичном уровне IPv4r2 изменений, возникших в новых редакциях относительно предыдущих редакций и мотивы принятия таких решений.

В редакции от 02 марта 2016

Исходный формат op3 в кодировании пользовательского адресного пространства имел следующие проблемы:

- <формат опции=001>[1] ошибочно имеет указанный размер 1 вместо 3;
- <формат опции=001> ошибочно закрывает будущее расширение кода опции op3;
- <формат опции=000> не следует правилу опций IPv4r2 "закрывать расширение кода опции единицей";

поэтому формат опции op3 в новой редакции описания протокола IPv4r2 изменяется.

В редакции от 31 декабря 2015

Предыдущее описание протокола TCPr2 содержало ошибку - поле размер окна не было расширено до 32 бит и не было вставлено резервное поле 16 бит. Поскольку протоколы UDPr2/TCPr2 продвинулись в своем логическом развитии в плане формализации адресации, формат заголовков этих протоколов в IPv4r2 полностью изменен в связи с появлением промежуточного протокола локальной адресации хоста HLAP.

Примеры реализации IPv4r2 протокола на реальных системах.

Примеры, если не указано обратное, приводятся для minix 3_1_0 (книжная версия). Эта версия бесплатна, имеет открытый код и доступна для скачивания по сети.

Нужно иметь в виду, что примеры 1 и 2 созданы до появления стандарта IPv4r2.0 и могут содержать отличия, примеры же реализации IPv4r2.0 совместимы с этой редакцией IPv4r2.

Использовать IPv4r2.0 для работы в интернете прямо сегодня и прямо сейчас.

Примеры реализации IPv4r2.0 позволяют для тестовых целей практически использовать minix в глобальных сетях IPv4r2 прямо сегодня и прямо сейчас, только прочитав описание стандарта IPv4r2.0 и просто установив книжный minix на глобальный IPv4 адрес как IPv4r2 клиент или IPv4r2 сервер, внеся в minix изменения из этих примеров.

Первый рабочий пример реализации протокола IPv4r2.

Для того чтобы реально показать разницу в сложности модификации программ с открытым исходным кодом при добавлении поддержки IPv4r2, по сравнению с добавлением поддержки IPv6, приведем пример программы, которая на minix машине выполняет тестовую отправку и прием IPv4r2 пакетов IPv4r2 адресации в режиме совместимости с IPv4.

Хотя я не имел опыта работы с сетью minix 3_1_0, такая же ситуация будет с каждым опытным пользователем, кто до этого пристально не разглядывал работу сетевой подсистемы своей машины, какого бы типа она не была.

Я сделал это улучшение для IPv4r2 в принципе за один день.

Сегодня 31 декабря 2015 года. Если завтра, 1 января 2016 года, ваш провайдер добавит IPv4r2.0 транслятор в свой шлюз

провайдера и каждому своему клиенту выделит IPv4r2 базовый адрес и свяжет такой адрес с IPv4 адресом клиента в локальной сети провайдера, то прямо с 1 января 2016 проблема исчерпания IPv4 адресов и адресации серверных ресурсов сети интернет будет решена.

Для того чтобы первый пример заработал, надо внести изменения в ряд файлов minix.

-) файл /usr/etc/rc

убрать dhcp

```
      #daemonize dhcpd
```

задать адрес и сразу после добавить конфигурацию и маршрут

```
      ifconfig -I /dev/ip0 -h 192.168.101.11 -n 255.255.255.0 -m 1152
      add_route -o -g 192.168.101.11 -d 169.254.0.0 -n 255.255.0.0 -m 1 -I /dev/ip0
```

-) файл /usr/src/servers/inet/generic/ip_lib.c

Следующий файл придется корректировать чтобы задать политику распознавания опций IPv4. Вообще то это связано не с IPv4r2, а с реализацией IPv4 в minix:

- в IPv4 заголовке есть только две опции однобайтового формата (0 и 1);
- а остальные опции в IPv4 заголовке должны быть двухбайтового формата, тогда IPv4 маршрутизаторы смогут их распознавать даже не зная их назначения, в этом смысл такого кодирования IPv4 опций;

в любом случае такое поведение IPv4 легко настроить, сами увидите:

1.

в функции

```
PUBLIC int ip_chk_hdropt (opt, optlen)
```

надо заменить

```
            default:
                  return EINVAL;
```

на

```
            default:
                  if (opt[1] < 2) return EINVAL;
                  i += opt[1];
                  opt += opt[1];
```

Вот и все модификации.

-) файл /usr/src/servers/inet/generic/ip_read.c

Корректировать этот файл приходится для задания правильной (по логике работы IPv4) маршрутизации IPv4 в minix, это

совсем не связано с IPv4r2, а только с реализацией IPv4 в minix, которая по ошибке пакеты из сетей типа 127.0.0.1 отправляет на входную маршрутизацию, так словно этот IPv4 пакет пришел из внешней сети.

(Далее в третьем примере файл ip_read.c приводится полностью)
1.
в функции
```
void ip_process_loopb(ev, arg)
      ip_arrived(ip_port, pack);
```
заменить на
```
      ip_loopb_arrived(ip_port, pack);
```

2.
добавить функцию ip_loopb_arrived

```
PUBLIC void ip_loopb_arrived(ip_port, pack)
ip_port_t *ip_port;
acc_t *pack;
{
      ip_hdr_t *ip_hdr;
      int ip_frag_len, ip_hdr_len;
      size_t pack_size;

      pack_size= bf_bufsize(pack);

      assert (pack_size >= IP_MIN_HDR_SIZE);
      pack= bf_align(pack, IP_MIN_HDR_SIZE, 4);
      pack= bf_packIffLess(pack, IP_MIN_HDR_SIZE);
      assert (pack->acc_length >= IP_MIN_HDR_SIZE);

      ip_hdr= (ip_hdr_t *)ptr2acc_data(pack);
      ip_hdr_len= (ip_hdr->ih_vers_ihl & IH_IHL_MASK) << 2;
      if (ip_hdr_len>IP_MIN_HDR_SIZE)
      {
            pack= bf_packIffLess(pack, ip_hdr_len);
            ip_hdr= (ip_hdr_t *)ptr2acc_data(pack);
      }
      ip_frag_len= ntohs(ip_hdr->ih_length);
```

```
      assert(ip_frag_len == pack_size);

      /* here local delivery only */
      ip_port_arrive (ip_port, pack, ip_hdr);
}
```

Я сделал ip_loopb_arrived из ip_arrived просто убрав все лишнее, насколько мог это лишнее распознать.

3.
в качестве комментария к реализации IPv4 в minix
в этой функции надо иметь в виду что NWIO_EN_BROAD это еще и "NWIO_EN_ROUTE"

```
PUBLIC void ip_port_arrive (ip_port, pack, ip_hdr)
      else
            ip_pack_stat= NWIO_EN_BROAD;
            /*
            in real here are the packets arrived by oroute or iroute tables
            assume NWIO_EN_ROUTE users also
            */
```

-) Сам тест (r2_nat.c).

Приведу полный листинг программы (на базе ping.c), которая посылает пакет на локальный адрес 169.254.0.0 вместе с IPv4r2 опциями для проверки того, что этот пакет проходит с опциями через сетевую подсистему IPv4 minix и может быть прочитан для NAT преобразования в удаленный адрес 192.168.101.12:
файл r2_nat.c:

```
/**/

#include <sys/types.h>
#include <errno.h>
#include <signal.h>
#include <net/gen/netdb.h>
#include <sys/ioctl.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <net/gen/oneCsum.h>
```

```
#include <fcntl.h>
#include <net/gen/in.h>
#include <net/gen/in4r2.h>
#include <net/gen/inet.h>
#include <net/gen/ip_hdr.h>
#include <net/gen/icmp_hdr.h>
#include <net/gen/ip_io.h>

#include <net/hton.h>
#include <string.h>

static u8_t i_buf[16*1024];
static u8_t o_buf[16*1024];

#define TSZ 240
static char sb[16];

#define where() fprintf(stderr, "%s %d:", __FILE__, __LINE__);

int main(argc, argv)
int argc;
char *argv[];
{
int                     fd, res, n;
nwio_ipopt_t      ipopt;

ip_hdr_t          *ip_hdr;
u8_t              *ip_opt_b45;
u8_t              *ip_opt_u12;
u8_t              *data;

int                     ps = 32; /* 20 + 8 + 4 */

FILE              *fo;

      fo=fopen("1","wb");
```

```
    if(!fo){ perror("open 1"); exit(1); }

    /**/
    fd= open ("/dev/ip0", O_RDWR);
    if (fd<0){ perror("open /dev/ip0"); exit(1); }

    /**/
    ipopt.nwio_flags=
          NWIO_COPY |NWIO_EN_LOC |NWIO_EN_BROAD |NWIO_REMANY
          |NWIO_PROTOANY |NWIO_RWDATALL
          |NWIO_HDR_O_ANY;

    res = ioctl (fd, NWIOSIPOPT, &ipopt);
    if (res<0){ perror("ioctl (NWIOSIPOPT)"); exit(1); }

    /**/
    ip_hdr = (ip_hdr_t*)o_buf;
    ip_opt_b45 = &o_buf[20];
    ip_opt_u12 = &o_buf[20 + 8];
    data = &o_buf[20 + 8 + 4];
    memset(o_buf,0,8);
    memset(ip_opt_b45,0,8);
    memset(ip_opt_u12,0,4);

    /**/
    ip_opt_b45[0] = IP_OPT_1;
    ip_opt_b45[1] = 8;
    ip_opt_b45[2] = 0x80;

    ip_opt_u12[0] = IP_OPT_3;
    ip_opt_u12[1] = 4;
    ip_opt_u12[2] = 0x10;

    /*
    ip_hdr -> ih_vers_ihl = 0x40;
    ?ip_hdr -> ih_id = 0;
```

```
	*/
	ip_hdr -> ih_tos = 0;
	ip_hdr -> ih_ttl = 32;
	ip_hdr -> ih_flags_fragoff = htons(IH_DONT_FRAG);

	/*ip_hdr -> ih_src = inet_addr("192.168.101.11");*/
	/*ip_hdr -> ih_dst = inet_addr("127.0.0.1");*/
	ip_hdr -> ih_dst = inet_addr("169.254.0.1");

	/*
	ip_hdr -> ih_proto = 0x11;
	ip_hdr -> ih_length = htons(TSZ);
	*/
	ip_hdr -> ih_vers_ihl |= ps/4;
	ip_hdr -> ih_proto = 0x0;

	/*
	ip_hdr -> ih_hdr_chk = 0;
	ip_hdr -> ih_hdr_chk = ~oneC_sum(0, (u16_t *)o_buf, ps);
	*/

	sprintf((char*)data, "%s", "test msg");
	fprintf(fo,"%-16s","orig post");
	fwrite(o_buf,1,TSZ,fo);
	fflush(fo);

	/**/
	res= write(fd, o_buf, TSZ);
	if (res<0){ perror("write"); exit(1); }
	if (res != TSZ)
	{
	      where();
	      fprintf(stderr, "write result= %d\n", res);
	      exit(1);
	}
```

```
      /**/
      alarm(5);
      for(n=0; n<10; ++n){

      res= read(fd, i_buf, sizeof(i_buf));
      if (res<0)
      {
            perror("read");
            exit(1);
      }

      where();
      fprintf(stderr, "read result= %u\n", res);

      sprintf(sb, "get %04u", n);
      fprintf(fo,"%-16s",sb);
      fwrite(i_buf,1,TSZ,fo);
      fflush(fo);
      }

      return 0;
}
```

-) Результат исполнения теста.

Вот результат работы этой программы на minix по передаче IPv4r2 пакета через IPv4 сетевой интерфейс minix:

```
0000000000: 6F 72 69 67 20 70 6F 73 | 74 20 20 20 20 20 20 20  orig post
0000000010: 08 00 00 00 00 00 40 00 | 20 00 00 00 00 00 00 00  ◘     @
0000000020: A9 FE 00 01 88 08 80 00 | 00 00 00 00 8B 04 10 00  ©ю ☺€◘Ђ     ‹♦►
0000000030: 74 65 73 74 20 6D 73 67 | 00 00 00 00 00 00 00 00  test msg
0000000040: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000050: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000060: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000070: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000080: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000090: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
```

```
00000000A0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000000B0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000000C0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000000D0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000000E0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000000F0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000100: 67 65 74 20 30 30 30 30 | 20 20 20 20 20 20 20 20  get 0000
0000000110: 48 00 00 F0 01 31 40 00 | 20 00 E3 1C C0 A8 65 0B  H  p☺1@   ┌└AËe♂
0000000120: A9 FE 00 01 88 08 80 00 | 00 00 00 00 8B 04 10 00  ©ю ☺€◘Ђ     ‹♦►
0000000130: 74 65 73 74 20 6D 73 67 | 00 00 00 00 00 00 00 00  test msg
0000000140: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000150: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000160: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000170: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000180: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
0000000190: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000001A0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000001B0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000001C0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000001D0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000001E0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
00000001F0: 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
```

По смещению 0x24 идут две опции IPv4r2 адресации (45 бит базовая 0x88 0x08) и по смещению 0x2C (12 бит пользовательская 0x8B 04). По флагам 0x80 и 0x10 эти IPv4r2 опции расширяют адрес назначения A9 FE 00 01 (169.254.0.1). Пакеты приходящие локально на этот адрес должны подвергнуться трансляции NAT (следующие примеры).

-) Тест NAT преобразования IPv4r2 адресации в локальные адреса IPv4.

Приведу и описание из предыдущих редакций IPv4r2 того, как сделать на базе первого примера NAT отображение IPv4r2 на IPv4. Это описание фактически проектное решение, которое и будет реализовано в следующих примерах, решение в котором нет особо ничего лишнего, что облегчает понимание.

Сделать NAT отображение IPv4r2 на IPv4 не сильно сложно, но в двух словах об этом не расскажешь, это будет многословно и непонятно, но выглядит это все в тестовом виде примерно так, файл nat_r2.c:

```
static nat_t           nat[NATS];

```

```
static init_nat_t      init_nat[]={
      {"169.254.0.1",{"192.168.101.12","",""}}
      ...
};

/*
"169.254.0.1" это локальный (по отношению к вашей IPv4 192.168.101.11 машине) IPv4 адрес
для удаленного (по отношению к 192.168.101.11) IPv4r2 адресата
"192.168.101.12/0.0.0.0/0.0.0.0/0.0.0.0:[порт]:[индекс шлюза]:0.0"
*/

Во время работы NAT трансляции записи в этой таблице можно менять перечитывая из /etc/nattab
   •  можно добавлять записи по SIGUSR1
   •  можно обновлять всю таблицу перераспределяя все адреса заново по SIGUSR2

...

      res= read(fd, i_buf, sizeof(i_buf));
      if (res<0){ perror("read"); exit(1); }

      ip_hdr = (ip_hdr_t*)i_buf;

      for(n=0; n<NATS; ++n){
            if(!nat[n].loc)break;
            if( nat[n].loc != ip_hdr->ih_dst )continue;

            res= do_dst_nat(n, res);
            if(res){
            if( write(fd, o_buf, res) != res ){ perror("write dst"); exit(1); }
            }
            break;
            }

...

В реализации do_nat нет ничего интересного, только в зависимости от протокола в поле заголовка IPv4 рутинный пересчет
```

```
контрольной суммы IPv4 и замена IPv4 адресов в этом заголовке.

int   do_dst_nat(n, b_size)
int n;
int b_size;
{
ip_hdr_t          *ip_hdr;

      ip_hdr = (ip_hdr_t*)i_buf;
      switch(ip_hdr->ih_proto){
      case 0x06:
            return ;

      case 0x11:
            return ;

      default:
            return ;
      }
}
```

-) Результаты тестов первого примера.

У приведенной тестовой реализации IPv4r2 на minix 3_1_0 конечно есть проблемы, но это, в общем то, проблемы реализации самого IPv4 на minix, независимые от IPv4r2.

Я добавил изменения нужные для IPv4r2 на систему, которую первый раз вижу, а вот если бы я захотел добавить на нее совместимость с IPv6 в таких же условиях, вот тут возникли бы трудности.

Именно для того чтобы избегать таких трудностей на стороне клиента и создан IPv4r2. А вот думать о том, как будут передаваться широко- адресные пакеты в сетях провайдера, другим участникам сети в общем то нет нужды, они даже не знают как работают те сети. Если провайдеру внутри своих скрытых от пользователя высокопроизводительных сетей нужна IPv6 адресация, то там она путь и будет, она не должна создавать проблемы конечным пользователям.

Второй рабочий пример реализации протокола IPv4r2.

В этом примере проводится сетевой обмен по протоколу IPv4r2 между двумя отдельными машинами по протоколу UDP.

-) Файл u_r2.c

Вот текст пользовательского IPv4r2 приложения u_r2.c, которое выполняет такой обмен, чтобы показать что с точки зрения приложения в IPv4r2 по отношению к IPv4 ничего особо не меняется:

```
/*
u_r2.c
*/

#define DEBUG     1

#include <sys/types.h>
#include <sys/ioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <signal.h>
#include <unistd.h>
#include <fcntl.h>
#include <net/gen/netdb.h>
#include <net/gen/oneCsum.h>
#include <net/gen/in.h>
#include <net/gen/inet.h>
#include <net/gen/ip_io.h>
#include <net/gen/ip_hdr.h>
#include <net/gen/udp.h>
#include <net/gen/udp_io.h>
#include <net/gen/udp_hdr.h>
#include <net/hton.h>

static u8_t       i_buf[16*1024];
static u8_t       o_buf[16*1024];

#define     IP_H_SIZE         20
#define     UDP_P_SIZE        64

/**/
```

```
void writeIpAddr(addr)
ipaddr_t addr;
{
#define addrInBytes ((u8_t *)&addr)

      fprintf(stdout, "%d.%d.%d.%d",
            addrInBytes[0], addrInBytes[1],
            addrInBytes[2], addrInBytes[3]
            );
#undef addrInBytes
}

/**/
int         main(argc, argv)
int         argc;
char  *argv[];
{
int                           res;

int                           ip_fd;
nwio_ipopt_t            ipopt;
ip_hdr_t                *ip_hdr[2];
udp_hdr_t               *udp_hdr[2];
u8_t                    *data[2];

unsigned                ip_size;
ipaddr_t                ip_own, ip_dst;
udpport_t               port_own, port_dst;

FILE                    *fo;

      fo=fopen("2","wb");
      if(!fo){ perror("fopen(\"2\",\"wb\")"); exit(1); }

      ip_fd= open("/dev/ip0",O_RDWR);
      if(ip_fd<0){ perror("open(\"/dev/ip0\",O_RDWR)"); exit(1); }
```

```
ipopt.nwio_flags =
      NWIO_COPY |NWIO_EN_LOC |NWIO_EN_BROAD |NWIO_REMANY
      |NWIO_PROTOANY |NWIO_RWDATALL
      |NWIO_HDR_O_ANY
      |NWIO_EN_ROUTE |NWIO_SRCAUTO |NWIO_IDAUTO |NWIO_HDRAUTO;

res= ioctl(ip_fd, NWIOSIPOPT, &ipopt);
if(res<0){ perror("ioctl(ip_fd, NWIOSIPOPT, &ipopt)"); exit(1); }

/**/
ip_hdr[0]= (ip_hdr_t*)i_buf;
ip_hdr[1]= (ip_hdr_t*)o_buf;

udp_hdr[0]= (udp_hdr_t*)(i_buf + IP_H_SIZE);
udp_hdr[1]= (udp_hdr_t*)(o_buf + IP_H_SIZE);

data[0]= i_buf + IP_H_SIZE + 8;
data[1]= o_buf + IP_H_SIZE + 8;

ip_own= inet_addr("192.168.101.11");
port_own= htons(0x8111);

ip_dst= inet_addr("169.254.0.12");
port_dst= htons(0x8112);

/**/
ip_hdr[1]->ih_vers_ihl = IP_H_SIZE/4;
ip_hdr[1]->ih_tos = 0;
ip_hdr[1]->ih_flags_fragoff = htons(IH_DONT_FRAG);
ip_hdr[1]->ih_ttl = 0x15;

ip_hdr[1]->ih_proto = 0x11;
ip_hdr[1]->ih_dst = ip_dst;

udp_hdr[1]->uh_src_port= port_own;
```

```
	udp_hdr[1]->uh_dst_port= port_dst;
	udp_hdr[1]->uh_length= htons(UDP_P_SIZE);
	udp_hdr[1]->uh_chksum= 0;

	/**/
	sprintf((char*)(data[1]),"%-16s","UDP msg req");

	/**/
	ip_size= UDP_P_SIZE+IP_H_SIZE;
	res= write(ip_fd, o_buf, ip_size);
	if( res!=ip_size ){ perror("write(ip_fd, o_buf, ip_size)"); exit(1); }

	/**/
	for(;;){
		alarm(60);
		res= read(ip_fd, i_buf, sizeof(i_buf));
		if( res<0 ){ perror("read(ip_fd, i_buf, sizeof(i_buf))"); exit(1); }
		alarm(0);

		if(
			( ip_hdr[0]->ih_src != ip_dst )
			||( ip_hdr[0]->ih_dst != ip_own )
			||( ip_hdr[0]->ih_proto != 0x11 )
			){

			fprintf(stdout, "\ru: drop answer 0x%x: ", ip_hdr[0]->ih_proto); writeIpAddr(ip_hdr[0]->ih_src);
			fprintf(stdout, " -> "); writeIpAddr(ip_hdr[0]->ih_dst);
			fprintf(stdout, "\n");
			continue;
		}

		if( fwrite(i_buf,128,1,fo) != 1){ perror("fwrite(i_buf,128,1,fo)"); exit(1); }
		fflush(fo);
		exit(0);
	}
}
```

-) Файл src/servers/inet/generic/ip_lib.c

Чтобы u_r2.c заработал, внесем пару улучшений в IPv4 реализацию minix.

Первое улучшение с точки зрения проверки опций, будем проверять что опции не выходят за размер памяти которая для них выделена, это небольшие изменения в функцию ip_chk_hdropt:

```
строка 35:
      while (i<optlen)
      {
            DBLOCK(2, printf("*opt= %d\n", *opt));

            /* single octet per option */
                  /* End of Option list */
            if( (*opt) == IP_OPT_EOL ) return NW_OK;
                  /* No Operation */
            if( (*opt) == IP_OPT_NOP ) { ++i; ++opt; continue; }

            /* at least 2 octets per option */
            if(
                  ( (i+2) > optlen )
                  ||( (i+opt[1]) > optlen )
                  ||( opt[1] < 2 )
            ){
                  DBLOCK(1, printf("unknown option\n"));
                  return EINVAL;
            }

строка 124:
            default:
                  i += opt[1];
                  opt += opt[1];
```

обратите внимание, что наш "default", изменения в который надо внести для поддержки опций IPv4r2, стал еще меньше, стал состоять только из двух строк, строка "if(opt[1]<2)return EINVAL;" переехала вверх, потому что эту проверку лучше выполнить для всех IPv4 опций.

-) Файл src/include/net/gen/ip_io.h

Второе улучшение внесем для того чтобы полнее управлять IP опциями в u_r2.c. Добавим пару опций контроля полей IP заголовка типа "NWIO_IDAUTO" для IP интерфейса minix, это относится только к IPv4 реализации протокола в minix, файл маленький, приведем его полностью:

```
/*
server/ip/gen/ip_io.h
*/

#ifndef __SERVER__IP__GEN__IP_IO_H__
#define __SERVER__IP__GEN__IP_IO_H__

typedef struct nwio_ipconf2
{
      u32_t nwic_flags;
      ipaddr_t nwic_ipaddr;
      ipaddr_t nwic_netmask;
} nwio_ipconf2_t;

typedef struct nwio_ipconf
{
      u32_t nwic_flags;
      ipaddr_t nwic_ipaddr;
      ipaddr_t nwic_netmask;
      u16_t nwic_mtu;
} nwio_ipconf_t;

#define NWIC_NOFLAGS          0x0
#define NWIC_FLAGS            0x7
#     define NWIC_IPADDR_SET        0x1
#     define NWIC_NETMASK_SET       0x2
#     define NWIC_MTU_SET           0x4

typedef struct nwio_ipopt
{
      u32_t nwio_flags;
      ipaddr_t nwio_rem;
```

```
	ip_hdropt_t nwio_hdropt;
	u8_t nwio_tos;
	u8_t nwio_ttl;
	u8_t nwio_df;
	ipproto_t nwio_proto;
} nwio_ipopt_t;

#define NWIO_NOFLAGS	0x0000l
#define NWIO_ACC_MASK	0x0003l
#	define NWIO_EXCL	0x00000001l
#	define NWIO_SHARED	0x00000002l
#	define NWIO_COPY	0x00000003l
#define NWIO_LOC_MASK	0x0010l
#	define NWIO_EN_LOC	0x00000010l
#	define NWIO_DI_LOC	0x00100000l
#define NWIO_BROAD_MASK	0x0020l
#	define NWIO_EN_BROAD	0x00000020l
#	define NWIO_DI_BROAD	0x00200000l
#define NWIO_REM_MASK	0x0100l
#	define NWIO_REMSPEC	0x00000100l
#	define NWIO_REMANY	0x01000000l
#define NWIO_PROTO_MASK	0x0200l
#	define NWIO_PROTOSPEC	0x00000200l
#	define NWIO_PROTOANY	0x02000000l
#define NWIO_HDR_O_MASK	0x0400l
#	define NWIO_HDR_O_SPEC	0x00000400l
#	define NWIO_HDR_O_ANY	0x04000000l
#define NWIO_RW_MASK	0x1000l
#	define NWIO_RWDATONLY	0x00001000l
#	define NWIO_RWDATALL	0x10000000l

/* !my */
 /* EN receive any dst packets by route tables */
#define NWIO_ROUTE_MASK	0x0040l
#	define NWIO_EN_ROUTE	0x00000040l
#	define NWIO_DI_ROUTE	0x00400000l
```

```
 /* AUTO ip_hdr.ih_src */
#define NWIO_SRC_MASK   0x2000l
#     define NWIO_SRCANY           0x00002000l
#     define NWIO_SRCAUTO          0x20000000l
 /* AUTO ip_hdr.ih_id, ip_hdr.fragoff */
#define NWIO_ID_MASK    0x4000l
#     define NWIO_IDANY       0x00004000l
#     define NWIO_IDAUTO           0x40000000l
 /* AUTO IP hdr */
#define NWIO_HDR_MASK   0x8000l
#     define NWIO_HDRANY           0x00008000l
#     define NWIO_HDRAUTO          0x80000000l

#endif /* __SERVER__IP__GEN__IP_IO_H__ */

/*
 * $PchId: ip_io.h,v 1.5 2001/03/12 22:17:25 philip Exp $
 */
```

Для того чтоб новые опции заработали, их конечно нужно реализовать в файле ip_write.c, но поскольку этот файл был еще раз модифицирован для IPv4r2.0, нет смысла приводить его промежуточные реализации, нужно взять самый последний измененный файл ip_write.c, который приведен в примерах далее.

-) Файл nat_r2.c

Ну и для того чтобы адрес "169.254.0.12" мог бы в этой программе u_r2.c быть проверен вот так "ip_hdr[0]->ih_dst != ip_own", при том что "ip_own= inet_addr("192.168.101.12");", надо выполнить NAT трансляцию IPv4 адресов "169.254.0.12" в "192.168.101.12".

Никто не будет оспаривать, что написать NAT трансляцию для IPv4 адресов можно, она применяется повсеместно. Для того чтобы как-то снизить нашу нагрузку по работе с IPv4 и minix и сосредоточиться на IPv4r2, применим упрощенную IPv4 трансляцию, которая работает как пользовательское приложение и подобно команде slip висит на /dev/ip и транслирует адреса по NAT таблице. Это и есть nat_r2.c, это прямая реализация того дизайна, что был анонсирован как NAT транслятор в предыдущем первом примере.

Приведем полный текст такой программы:

```
#include <net/hton.h>
```

```
#include <string.h>

#include "inat.h"

#define IPV4R2_OPT_SIZE 12

static nat_t nat[NATS+1];
static u32_t nat_items;

static init_nat_t init_nat[]={
      {"169.254.0.1", {"192.168.101.1" ,"",""}},
      {"169.254.0.11",{"192.168.101.11","",""}},
      {"169.254.0.12",{"192.168.101.12","",""}}
};

static u8_t i_buf[16*1024];
static u8_t o_buf[16*1024];

/*
in general ip_nat IPv4 must not be the same as ip_nat_local (local NAT user IPv4)
otherwise NAT local IPs must not directly connect to the remote IPs (nat[n].dst.ipv4)
only to local IPs (nat[n].loc)
for example:
- ip_nat.d = ip_nat_local.d+100 (reserves separated local IP for NAT in the same local net)
- ip_nat and ip_nat_local are connected to separated /dev/ipX
*/
static ipaddr_t   ip_nat;
static ipaddr_t   ip_nat_local;

#define where() fprintf(stderr, "%s %d:", __FILE__, __LINE__);

/**/
void writeIpAddr(addr)
ipaddr_t addr;
{
#define addrInBytes ((u8_t *)&addr)
```

```
      fprintf(stderr, "%d.%d.%d.%d", addrInBytes[0], addrInBytes[1],
            addrInBytes[2], addrInBytes[3]);
#undef addrInBytes
}

/**/
int   do_dst_nat(n)
unsigned n;
{
ip_hdr_t          *ip_hdr;
u8_t              *ip_hdr_opt;
unsigned          ip_size;

ipaddr_t          old_ih_dst, old_ih_src;
unsigned          old_ih_size, old_opt_size;
unsigned          ih_size, opt_tail_size, data_size;
unsigned          tmp;

ps_hdr_tcp_t      ps_hdr_tcp;
U16_t             *ps_hdr_chk;

      ip_hdr = (ip_hdr_t*)i_buf;

      old_ih_src = ip_hdr->ih_src;
      ip_hdr->ih_src = ip_nat;

      old_ih_dst = ip_hdr->ih_dst;
      ip_hdr->ih_dst = nat[n].dst.ipv4;

      #if 0
      /* add new ip header aligned options */
      ip_hdr_opt = o_buf + sizeof(ip_hdr_t);
      memcpy(ip_hdr_opt, nat[n].dst.ip_opt_b45, 8);
      memcpy(ip_hdr_opt+8, nat[n].dst.ip_opt_u12, 4);
      ip_hdr_opt += IPV4R2_OPT_SIZE;
```

```
/*check old ip options here (must not be IPv4r2 dst opt) */

/*merge old ip options */
old_ih_size = (ip_hdr->ih_vers_ihl & IH_IHL_MASK)*4;
old_opt_size = old_ih_size - sizeof(ip_hdr_t);
if(old_opt_size){
      memcpy(ip_hdr_opt, i_buf+sizeof(ip_hdr_t), old_opt_size);
      ip_hdr_opt += old_opt_size;
      }

/*check ip header align*/
ih_size = old_ih_size + IPV4R2_OPT_SIZE;
/* aleady must be
opt_tail_size= ih_size % 4;
if(opt_tail_size){
      tmp = 4-opt_tail_size;
      memset(ip_hdr_opt, 0, tmp);
      ip_hdr_opt += tmp;
      ih_size += tmp;
      }
*/

/*check ip header size */
if( ih_size > (15*4) ){
      fprintf(stderr, "IPv4r2 NAT: %s\n", "IP header too big");
      return 0;
      }

/*merge data*/
data_size = NTOHS(ip_hdr->ih_length) - old_ih_size;
memcpy(ip_hdr_opt, i_buf+old_ih_size, data_size);

/**/
ip_size = ih_size + data_size;
ip_hdr->ih_length = HTONS(ip_size);
```

```
ip_hdr->ih_vers_ihl |= (ih_size)/4;

/*
ip_hdr->ih_id=0;
ip_hdr->ih_flags_fragoff = HTONS(IH_DONT_FRAG);
*/

ip_hdr->ih_hdr_chk = 0;
/*
ip_hdr->ih_hdr_chk = ~oneC_sum(0, (u16_t *)i_buf, ih_size);
*/
memcpy(o_buf,i_buf,sizeof(ip_hdr_t));

#else
ip_size= NTOHS(ip_hdr->ih_length);
ih_size= (ip_hdr->ih_vers_ihl & IH_IHL_MASK)*4;
ip_hdr_opt = o_buf+ih_size;

ip_hdr->ih_hdr_chk = 0;
memcpy(o_buf,i_buf,ip_size);
#endif

/* chksum */
switch(ip_hdr->ih_proto){
case 0x06:
      /* TCP chksum */
      memset( &ps_hdr_tcp, 0, sizeof(ps_hdr_tcp) );
      ps_hdr_tcp.not_old_ih_src= ~old_ih_src;
      ps_hdr_tcp.not_old_ih_dst= ~old_ih_dst;
      ps_hdr_tcp.new_ih_src= ip_hdr->ih_src;
      ps_hdr_tcp.new_ih_dst= ip_hdr->ih_dst;

      ps_hdr_chk = (U16_t*)(ip_hdr_opt+TCP_HDR_CHK);
      (*ps_hdr_chk) = ~oneC_sum(~(*ps_hdr_chk), &ps_hdr_tcp, sizeof(ps_hdr_tcp));
      return ip_size;
```

```
      case 0x11:
            /* dis UDP chksum */
            memset(ip_hdr_opt+UDP_HDR_CHK, 0, 2);
            return ip_size;

      default:
            /* assume own chksum */
            return ip_size;
      }
}

int   do_src_nat(n)
unsigned n;
{
ip_hdr_t          *ip_hdr;
u8_t              *ip_hdr_opt;
unsigned          ip_size;

ipaddr_t          old_ih_dst, old_ih_src;
unsigned          ih_size;

ps_hdr_tcp_t      ps_hdr_tcp;
U16_t             *ps_hdr_chk;

      ip_hdr = (ip_hdr_t*)i_buf;

      old_ih_src = ip_hdr->ih_src;
      ip_hdr->ih_src = nat[n].loc;

      old_ih_dst = ip_hdr->ih_dst;
      ip_hdr->ih_dst = ip_nat_local;

      #if 0
      /* skip IPv4r2 options here */
```

```
	/**/
	ip_size = NTOHS(ip_hdr->ih_length);
	ih_size = (ip_hdr->ih_vers_ihl & IH_IHL_MASK)*4;
	ip_hdr_opt = o_buf+ih_size;

	/**/
	ip_hdr->ih_length = HTONS(ip_size);

	ip_hdr->ih_vers_ihl |= (ih_size)/4;
	/*
	ip_hdr->ih_id=0;
	ip_hdr->ih_flags_fragoff = HTONS(IH_DONT_FRAG);
	*/

	ip_hdr->ih_hdr_chk = 0;
	/*
	ip_hdr->ih_hdr_chk = ~oneC_sum(0, (u16_t *)i_buf, ih_size);
	*/
	memcpy(o_buf,i_buf,ip_size);

	#else
	ip_size= NTOHS(ip_hdr->ih_length);
	ih_size= (ip_hdr->ih_vers_ihl & IH_IHL_MASK)*4;
	ip_hdr_opt = o_buf+ih_size;

	ip_hdr->ih_hdr_chk = 0;
	memcpy(o_buf,i_buf,ip_size);
	#endif

	/* chksum */
	switch(ip_hdr->ih_proto){
	case 0x06:
	      /* TCP chksum */
	      memset( &ps_hdr_tcp, 0, sizeof(ps_hdr_tcp) );
	      ps_hdr_tcp.not_old_ih_src= ~old_ih_src;
	      ps_hdr_tcp.not_old_ih_dst= ~old_ih_dst;
```

```
            ps_hdr_tcp.new_ih_src= ip_hdr->ih_src;
            ps_hdr_tcp.new_ih_dst= ip_hdr->ih_dst;

            ps_hdr_chk = (U16_t*)(ip_hdr_opt+TCP_HDR_CHK);
            (*ps_hdr_chk) = ~oneC_sum(~(*ps_hdr_chk), &ps_hdr_tcp, sizeof(ps_hdr_tcp));
            return ip_size;

      case 0x11:
            /* dis UDP chksum */
            memset(ip_hdr_opt+UDP_HDR_CHK, 0, 2);
            return ip_size;

      default:
            /* assume old chksum */
            return ip_size;
      }
}

/**/
char  chk_ip_frame( ip_hdr, b_size )
ip_hdr_t    *ip_hdr;
unsigned    b_size;
{
      /* check IP packet size */
      if( NTOHS(ip_hdr->ih_length) != b_size ){
      fprintf(stderr, "IPv4r2 NAT: %s\n", "read wrong IP header length");
      return 0;
      }

      /* check frag flag */
      /*
      if( !(ip_hdr->ih_flags_fragoff & HTONS(IH_DONT_FRAG)) ){
      fprintf(stderr, "IPv4r2 NAT: %s\n", "read IP fragments are not supported");
      return 0;
      }
      */
```

```
	return 1;
}

/**/
int main(argc, argv)
int argc;
char *argv[];
{
int			fd, res;
unsigned		n;
nwio_ipconf_t	ipconf;

nwio_ipopt_t	ipopt;
ip_hdr_t		*ip_hdr;

ipaddr_t		tmp;
icmp_hdr_t		*icmp_hdr;

	/* NAT translate table */
	memset(nat, 0, sizeof(nat));
	for(n=0; n<sizeof(init_nat)/sizeof(init_nat_t); ++n){
		if( n >= NATS)break;
		nat[n].loc = inet_addr( init_nat[n].loc );
		nat[n].dst.ipv4 = inet_addr( init_nat[n].dst.ipv4 );
		/*
		nat[n].dst.ip_opt_b45 =
		nat[n].dst.ip_opt_u12 =
		*/
		fprintf(stderr, "%02u: ", n); writeIpAddr(nat[n].loc);
		fprintf(stderr, " ", n); writeIpAddr(nat[n].dst.ipv4);
		fprintf(stderr, "\n");
	}

	/* NAT IP interface (external) */
	fd= open ("/dev/ip0", O_RDWR);
```

```
	if (fd<0){ perror("open /dev/ip0"); exit(1); }

	/**/
	res= ioctl (fd, NWIOGIPCONF, &ipconf);
	if (res < 0){
		fprintf(stderr, "Unable to get IP configuration: %s\n",    strerror(errno));
		exit(1);
	}
	ip_nat = ipconf.nwic_ipaddr;

	/**/
	ipopt.nwio_flags=
		NWIO_COPY |NWIO_EN_LOC |NWIO_EN_BROAD |NWIO_REMANY
		|NWIO_PROTOANY |NWIO_RWDATALL
		|NWIO_HDR_O_ANY
		|NWIO_EN_ROUTE |NWIO_SRCANY |NWIO_IDANY |NWIO_HDRAUTO;

	res = ioctl (fd, NWIOSIPOPT, &ipopt);
	if (res<0){ perror("ioctl (NWIOSIPOPT)"); exit(1); }

	/* NAT local IP interface */
	ip_nat_local = ip_nat;

	fprintf(stderr, "IP NAT: "); writeIpAddr(ip_nat);
	fprintf(stderr, "; IP NAT local: "); writeIpAddr(ip_nat_local);
	fprintf(stderr, "\n");

	/**/
	for(;;){

	res= read(fd, i_buf, sizeof(i_buf));
	if (res<0){ perror("read"); exit(1); }

	ip_hdr = (ip_hdr_t*)i_buf;

	/* check IP packet version */
```

```
	if( (ip_hdr->ih_vers_ihl >>4) != 4 ){
		fprintf(stderr, "IPv4r2 NAT: %s: 0x%x\n", "read wrong IP header version", ip_hdr->ih_vers_ihl );
		continue;
		}

	#if 1
	fprintf(stderr, "NAT got 0x%x: ", ip_hdr->ih_proto); writeIpAddr(ip_hdr->ih_src);
	fprintf(stderr, " to "); writeIpAddr(ip_hdr->ih_dst);
	fprintf(stderr, " id 0x%x", ip_hdr->ih_id);
	fprintf(stderr, "\n");

	if(ip_hdr->ih_proto == IPPROTO_ICMP){
		icmp_hdr= (icmp_hdr_t*)(i_buf+(ip_hdr->ih_vers_ihl & IH_IHL_MASK)*4);
		fprintf(stdout, "  [icmp: type 0x%02x; code 0x%02x]\n", icmp_hdr->ih_type, icmp_hdr->ih_code );
		}
	#endif

	for(n=0; n<NATS; ++n){
		if(!nat[n].loc){
			#if 0
			fprintf(stderr, "NAT exit %02u: ", n); writeIpAddr(ip_hdr->ih_src);
			fprintf(stderr, " to "); writeIpAddr(ip_hdr->ih_dst);
			fprintf(stderr, " ("); writeIpAddr(nat[n].loc);
			fprintf(stderr, ")\n");
			#endif
			break;
		}
		#if 1
		fprintf(stderr, "NAT proc %02u: loc ", n); writeIpAddr(nat[n].loc);
		fprintf(stderr, ", dst "); writeIpAddr(nat[n].dst.ipv4);
		fprintf(stderr, "\n");
		#endif

		/**/
		if( nat[n].loc == ip_hdr->ih_dst ){
		tmp= ip_hdr->ih_src;
```

```
#if 1
fprintf(stderr, "NAT dst: "); writeIpAddr(nat[n].loc);
fprintf(stderr, " -> "); writeIpAddr(nat[n].dst.ipv4);
fprintf(stderr, "\n\t(src: "); writeIpAddr(tmp);
fprintf(stderr, " -> ");  fprintf(stderr, "IN "); writeIpAddr(ip_nat);
fprintf(stderr, ")\n");
#endif

if(! chk_ip_frame( ip_hdr, res ) )break;

res= do_dst_nat(n);
if(res){ if( write(fd, o_buf, res) != res ){ perror("write dst"); } }
break;
}

/**/
if( nat[n].dst.ipv4 == ip_hdr->ih_src ){
tmp= ip_hdr->ih_dst;

if(
      ( ip_hdr->ih_dst != ip_nat ) /* not for NAT */
      ||( nat[n].dst.ipv4 == ip_nat_local ) /* bad NAT table */
      ){
      #if 1
      fprintf(stderr, "NAT src: drop "); writeIpAddr(ip_hdr->ih_src);
      fprintf(stderr, " to "); writeIpAddr(ip_hdr->ih_dst);
      fprintf(stderr, ")\n");
      #endif
      continue;
      }

#if 1
fprintf(stderr, "NAT src: "); writeIpAddr(nat[n].dst.ipv4);
fprintf(stderr, " -> "); writeIpAddr(nat[n].loc);
fprintf(stderr, "\n\t(dst: "); writeIpAddr(tmp);
```

```
            fprintf(stderr, " -> "); fprintf(stderr, "INL "); writeIpAddr(ip_nat_local);
            fprintf(stderr, ")\n");
            #endif

            if(! chk_ip_frame( ip_hdr, res ) )break;

            res= do_src_nat(n);
            if(res){ if( write(fd, o_buf, res) != res ){ perror("write src"); } }
            break;
            }

            /* for NATS records */
            }

      }
}
```

Этот метод NAT трансляции адресов IPv4r2 в IPv4 реально работает, вот UDP пакет в ответ на UDP запрос, оба пакета доставлены уже между двумя разными машинами в сети через такой NAT транслятор:

```
0000000000: 45 00 00 54 00 84 40 00 | 15 11 95 57 A9 FE 00 0C  E  T „@ §◄•W©ю ♀
0000000010: C0 A8 65 0B 81 12 81 11 | 00 40 00 00 55 44 50 20  AËe♂Ґ↕Ґ◄ @  UDP
0000000020: 6D 73 67 20 72 65 70 6C | 79 20 20 20 00 00 00 00  msg reply
```

И любое приложение, которое как минимум проверяет ему ли адресованы пакеты (а пакеты, которые адресованы другим просто игнорируются), работает с этой NAT системой нормально. Но ряд приложений с такой подменой адресов на самой машине все же не работают.

Некоторые из них не будут работать если иные машины в сети просто случайно обратятся к их ресурсам, поскольку их протокол разрушается если они принимают неправильные запросы или ответы от разных машин в сети. Другие откажутся работать в случае, когда в машине просто установлено более одной сетевой карты, поскольку они не знают какое устройство и когда надо открывать. В дополнение ко всему они прослушивают порты с адресом совпадающим с NAT.

Особенно много таких приложений навалилось на пользователей во времена когда была Windows XP. По хорошему для них в ОС должна быть поддержана виртуальная сетевая среда, где есть единственный виртуальный сетевой адаптер, который уже маршрутизирует пакеты внутри самой машины по потребности, а реальные устройства и подробности сетевой организации машины

должны быть от таких приложений скрыты.

Все эти проблемы не относятся к протоколу IPv4r2 и такие программы не будут работать даже с оригинальным IPv4 и NAT трансляцией, но нам чтобы решить проблему таких приложений для практической работы с IPv4r2, можно использовать фильтр, который встанет в цепочку между IP сервисом машины и сетевой картой и будет там осуществлять NAT трансляцию.

Практическая реализация такой вставки зависит от системы. Вот это и рассмотрим в следующих примерах.

Третий рабочий пример, реализация протокола "IPv4r2.0".

minix, был выбран для иллюстрации работы протокола, потому что он мал (это важнейшее качество при той документации что поставляется для программ с открытым кодом), не претендует на работу с любой аппаратурой (и не пытается опираться в этом деле на самого базового уровня нерешенные проблемы языка и дизайна, без которых задача просто не решится хорошо), имеет улучшенную внутреннюю структуру, имеет некоторые описания дизайна, совместим с некоторыми важными приложениями, бесплатен и доступен всем для скачивания и установки.

Для для NAT сервиса и для работы с сетевыми службами minix лучше было бы создать отдельный драйвер (аналог minix psip), который с одной стороны, полностью отвечает интерфейсу dev/ip, а с другой стороны, полностью отвечает интерфейсу аналогичному dev/psip (если бы в minix сам psip работал как хотелось бы, то не было бы необходимости делать новый psip).

Такой способ более предпочтителен чем жесткое встраивание NAT в код IP сервиса и в целом отличается от внешнего шлюза для доступа в интернет из локальной сети только тем, что код такого шлюза работает на самой локальной машине.

Проблемы, которые надо решать именно в minix при работе с psip, это:

- особенности взаимодействия между процессами в minix и передачи данных между ними;
- совместимость с уже готовым ПО minix, которое полагается на формат сетевого интерфейса minix;

и эти проблемы не имеют отношения к IPv4r2.

Тестовое приложение, написанное в соответствии с правилами IPv4r2, работало на minix в общем уже с 1 января, передавая по протоколу UDP данные между двумя машинами в сети с помощью IPv4r2 адресов, но надо обеспечить работу того ПО, что уже есть в minix.

Вы уже должно быть заметили, что особенностью minix является то, что интерфейсы работы с сетью в minix специальные, уникальные и все приложения опираются на этот интерфейс прямо, без промежуточных сервисов, типа posix, этот вопрос реализации сети в minix вообще не имеет отношения к IPv4r2, поэтому в целях иллюстрации работы самого IPv4r2 легче всего встроить локальную NAT трансляцию прямо в IP сервис minix (в лучших традициях худшего стиля, сочетающего плохо структурированный код на C и монолитные системы), подобно тому, как была исправлена функция minix проверки параметров

IPv4 заголовка:

```
      default:
            i += opt[1];
            opt += opt[1];
```

Тогда здесь можно будет привести только код NAT трансляции нужный только для IPv4r2 и это все будет работоспособным практически, что важно.

Но даже этот метод приводит к большим, а уж для целей публикации здесь точно большим, изменениям в коде. Код выглядит страшным, но на самом деле он ничего не делает, только переливает из пустого в порожнее. Это, скажем так, особенности конкретных программных интерфейсов уже существующих в minix, которые плохо разделяют интерфейс и реализацию, плохо документированы и всем этим досаждают программистам, решающим частные задачи и не позволяют легко воспользоваться преимуществами многоуровневой организации сети, не зависящей от системы. В тех системах, где эти интерфейсы описаны и проработаны лучше, задача поддержки IPv4r2 решится намного проще.

Число файлов, которые к 2 февраля 2016 уже надо было:

- модифицировать очень просто, достигло 3;
- модифицировать сложнее, достигло 3;
- добавить, достигло 2,
- плюс еще 3 модифицированных файла.

Это неприемлемо сложно по объему и по реализации.

У любой системы есть границы сложности, которые обусловлены ее исходным дизайном и с имеющимся дизайном интерфейсов сетевых служб minix такие модификации для NAT эта система не выдерживает сама по себе, даже если их проводить в пределах только исходного IPv4, а из того что уже есть для minix легче всего сделать NAT транслятор из отдельной выделенной машины minix, так же как уже работает скажем SLIP в этом же minix, это позволит сосредоточиться на проблемах IPv4r2.

В рамках выделенной minix машины для NAT, если не пытаться, как сделано в этом примере, на одной и той же машине minix запустить еще и NAT транслятор, модификации в исходный код minix для поддержки IPv4r2 будут иметь почти вот такой вид по сложности

```
      default:
            i += opt[1];
            opt += opt[1];
```

но практическая польза от выделенной minix машины намного меньше, чем одна minix машина как клиент или сервер или шлюз, а практическая польза это решающий фактор и можно пойти на жертвы в виде захламления сетевого сервиса minix, тем более что он все равно нуждается в реструктуризации.

Естественно, что сам алгоритм NAT трансляции имеет некий объем кода, но это не относится к IPv4r2, этот код всегда будет, независимо от версии IP, которая транслируется маршрутизатором.

Также NAT транслятор и UDP инкапсуляция будут необходимы любому расширению протокола, что IPv4r2, что IPv6, что еще какому либо, но все остальные улучшения для IPv4r2 проще, чем для IPv6.

Далее приведем листинг тестового, но работающего IPv4r2 NAT транслятора, который уже практически позволяет установить minix как IPv4r2 сервер пригодный для доступа к нему через интернет по IPv4r2 или использовать minix как клиент для доступа к другим IPv4r2 серверам в интернет.

Список модифицированных файлов:

- src/include/net/gen/ip_io.h (от предыдущих примеров)
- src/servers/inet/generic/ip_lib.c (от предыдущих примеров)
- src/servers/inet/generic/ip_nat.c
- src/servers/inet/generic/ip_nat_cfg.c
- src/servers/inet/generic/ip_nat.h
- src/servers/inet/generic/ip_write.c
- src/servers/inet/generic/ip_read.c
- src/servers/inet/generic/ip.c
- src/servers/inet/generic/ipr.c
- src/servers/inet/generic/ipr.h
- src/servers/inet/generic/ip_int.h

.

-) Файл src/servers/inet/generic/ip_nat.c

Этот файл содержит код NAT транслятора IPv4r2, этого файла нет в исходной сборке minix.

```
/*
ip_nat.c
*/

#include "inet.h"
```

```
#include "buf.h"
#include "event.h"
#include "type.h"

#include "arp.h"
#include "assert.h"
#include "clock.h"
#include "eth.h"
#include "icmp_lib.h"
#include "io.h"
#include "ip.h"
#include "ip_int.h"
#include "ipr.h"

#include "tcp.h"
#include "udp.h"

#include <net/gen/inet.h>
#include "ip_nat.h"

THIS_FILE

/* debug verbouse information */
#define DVI_I 1
#define DVI_W 1
#define DVI_R 1

PUBLIC char parse_url(ipaddr_t *dst, const char *src_str, unsigned level);
PUBLIC void ip_nat_print_hex(const void *buf, unsigned len, unsigned spline);
```

```
PUBLIC void ip_nat_init(void);
PUBLIC acc_t *ip_nat_read(ip_port_t *ip_port, acc_t *data);
PUBLIC acc_t *ip_nat_write(ip_port_t *ip_port, acc_t *data);

/* NAT working mode static parameters adjustment */
#include "ip_nat_cfg.c"

/* ********************************************************** */
/* NAT translate table */
static ip_nat_t         ip_nat[IP_NAT_ITEMS];
static u32_t       ip_nat_items;

static ipaddr_t         mapped_own_IPv4;
static ipaddr_t         own_IPv4;
static ip_nat_dst_t     own_IPv4r2;
static ipaddr_t         IPv4r2_src;

static u8_t             ip_opt_gc[4];

static ipaddr_t         IPv4r2_UDP_dst;
static udpport_t   IPv4r2_UDP_port;
static ipaddr_t         IPv4r2_UDP_uni_dst;

/**/
FORWARD void mk_b45(u8_t *dst, const char *src_str);
FORWARD void mk_u12(u8_t *dst, const char *src_str);
FORWARD void mk_v6(u8_t *dst, const char *src_str);
FORWARD void mk_gc(u8_t *dst, u8_t cmd);
FORWARD void mk_init_loc(ipaddr_t *loc, const char *const *init_loc_pp );
```

```
FORWARD void mk_init_dst(ip_nat_dst_t *ip_nat_dst, init_ip_nat_dst_t *init_ip_nat_dst);
FORWARD void mk_init(ip_nat_t *ip_nat, init_ip_nat_t *init_ip_nat);

/**/
static void mk_b45( dst, src_str )
u8_t     *dst;
const char    *src_str;
#if 1
{
ipaddr_t tmp;
u8_t     *buf;

     dst[0]= IP_OPT_R2_1;
     dst[1]= 8;
     memset(dst+2, 0, dst[1]-2);

     /* IPv4r2 set addr */
     #if 0
     /* only low byte (values d4 "0..255" accepted only) */
     dst[7]=strtoul(src_str,0,10);
     #else
     buf= (u8_t *)&tmp;

     if( parse_url(&tmp, src_str, 0) ){
          tmp= ntohl(tmp);
          dst[7]= buf[0];
          dst[6]= buf[1];
     }
     if( parse_url(&tmp, src_str, 1) ){
```

```
          tmp= ntohl(tmp);
          dst[5]= buf[0];
          dst[4]= buf[1];
          dst[3]= buf[2];
          dst[2]|= buf[3] & 0x03;
     }
     if( parse_url(&tmp, src_str, 2) ){
          tmp= ntohl(tmp);
          dst[2]|= (buf[0] & 0x07) << 2;
     }
     #endif

     #if DVI_I > 1
     printf("b45: ");
     ip_nat_print_hex(dst, dst[1], 16);
     #endif

     /* IPv4r2 clear flags */
     /* <flag>[3]<addr>[5] */
     dst[2] &= 0x1f;
}
#endif

static void mk_u12( dst, src_str )
u8_t      *dst;
const char    *src_str;
#if 1
{
ipaddr_t tmp;
```

```
u8_t     *buf;

    dst[0]= IP_OPT_R2_3;
    dst[1]= 4;
    memset(dst+2, 0, dst[1]-2);

    /* IPv4r2 set addr */
    #if 0
    /* skip high halfbyte (values "0..255" accepted only) */
    dst[3]= strtoul(src_str,0,10) & 0x0ff;
    #else
    buf= (u8_t *)&tmp;

    if( parse_url(&tmp, src_str, 0) ){
         tmp= ntohl(tmp);
         dst[3]= buf[0];
         dst[2]|= buf[1] & 0x0f;
    }
    #endif

    #if DVI_I > 1
    printf("u12: ");
    ip_nat_print_hex(dst, dst[1], 16);
    #endif

    /* IPv4r2 clear flags, set option */
    /* <option=000>[3]<flag>[1]<high byte>[4] */
    dst[2] &= 0x0f;
    dst[2] |= 0x20;
```

```
}
#endif

/**/
static void mk_v6( dst, src_str )
u8_t     *dst;
const char    *src_str;
#if 1
{
ipaddr_t tmp;
unsigned n;

     dst[0]= IP_OPT_R2_1;
     dst[1]= 20;
     memset(dst+2, 0, dst[1]-2);

     /* IPv4r2 set addr */
     #if 1
     for(n=0; n<4; ++n){
     if( parse_url(&tmp, src_str, n) ){
          memcpy(
               dst+(4+n*4),
               &tmp,
               4
               );
     }}
     #endif

     #if DVI_I > 1
```

```
     printf("v6: ");
     ip_nat_print_hex(dst, dst[1], 16);
     #endif

     /* IPv4r2 clear flags */
     /* <flag>[1]<0>[15] */
     dst[2] &= 0x00;
     dst[3] &= 0x00;
}
#endif

/**/
FORWARD void mk_gc(u8_t *dst, u8_t cmd)
#if 1
{
     dst[0]= IP_OPT_R2_3;
     dst[1]= 3;
     dst[2]= cmd & 0x0f;

     dst[2]&= 0x0f;
     dst[2]|= 0x10;

     /* default align, is not mandatory */
     dst[3]= 0x01;
}
#endif

/**/
static void mk_init_loc(loc, init_loc_pp)
```

```
ipaddr_t *loc;
const char    *const *init_loc_pp;
#if 1
{
     const char *init_loc= *init_loc_pp;

     if( strlen(init_loc) ){
          *loc = inet_addr( init_loc );
     }
}
#endif

static void mk_init_dst(ip_nat_dst, init_ip_nat_dst)
ip_nat_dst_t       *ip_nat_dst;
init_ip_nat_dst_t  *init_ip_nat_dst;
#if 1
{
     if( strlen(init_ip_nat_dst->ipv4) ){
          ip_nat_dst->ipv4 = inet_addr( init_ip_nat_dst->ipv4 );
     }

     if( strlen(init_ip_nat_dst->ip_opt_b45) ){
          ip_nat_dst->ip_opt_flags|= IP_NAT_R2_EN_B45;
          mk_b45( ip_nat_dst->b.ip_opt_b45, init_ip_nat_dst->ip_opt_b45 );
     }

     if( strlen(init_ip_nat_dst->ip_opt_u12) ){
          ip_nat_dst->ip_opt_flags|= IP_NAT_R2_EN_U12;
          mk_u12( ip_nat_dst->u.ip_opt_u12, init_ip_nat_dst->ip_opt_u12 );
```

```
    }

    if( strlen(init_ip_nat_dst->ip_opt_v6) ){
        ip_nat_dst->ip_opt_flags|= IP_NAT_R2_EN_V6;
        mk_v6( ip_nat_dst->v.ip_opt_v6, init_ip_nat_dst->ip_opt_v6 );
    }
}
#endif

static void mk_init(ip_nat, init_ip_nat)
ip_nat_t      *ip_nat;
init_ip_nat_t *init_ip_nat;
#if 1
{
    mk_init_loc(&ip_nat->loc, &init_ip_nat->loc);
    mk_init_dst(&ip_nat->dst, &init_ip_nat->dst);
}
#endif

/* ********************************************************** */
/*
return 0 if level not found or inet_addr failed
level 0,1,2 ... for ip0/ip1/ip2 ...
*/
char parse_url(dst, src_str, level)
ipaddr_t *dst;
const char    *src_str;
unsigned level;
#if 1
```

```
{
unsigned b, n, len, bs;
u8_t     buf[256];

u8_t     *point, *start;
unsigned points;
ipaddr_t tmp;
char     res;

    /* chk params */
    #if 1
    if( !dst || !src_str )return 0;

    bs= sizeof(buf);
    if(!bs)return 0;
    bs-= 1;

    len= strlen(src_str);
    if(len>bs)len= bs;
    if(!len)return 0;

    #endif

    /* find level */
    #if 1
    b=0;
    for( n=0; n<len; ++n )
    {
        if(src_str[n] != '/')continue;
```

```
        if( !(n-b) )return 0;

        if(!level)break;
        level-= 1;
        b= n+1;
    }
    if( level )return 0;
    if( b >= len )return 0;
    #endif

    /**/
    len= n-b;
    if(!len)return 0;

    memcpy(  buf, &src_str[b], len );
    buf[len]= 0;

    /*printf("MD1 buf: %s\n",buf);*/

    /* "low bytes" IPv4r2 addr format */
    #if 1
    points=0;
    for(n=0; n<len; ++n){ if( buf[n] == '.' )++points; }

    switch(points){
    case 0:
    case 3:
    #if 1
    /*
```

```
    IPv4 accepted:
         addr aligned to low (to dX)
              *dst= htonl( atol(buf) ), return 1;
         and byte.byte.byte.byte
    */
         return inet_aton((const char*)buf, dst);
    #endif

    case 1:
    case 2:
    #if 1
    /*
    aligned to low (to dX)
         byte.byte
         byte.byte.byte
    */
         *dst= 0;
         start= buf;
         for(n=0; n<=points; ++n){

         if(start){ point= (u8_t*)strchr((const char*)start,'.'); }
         if(point){ *point++ = 0; }

         res= inet_aton((const char*)start, &tmp);
         if(!res)return 0;
         tmp= ntohl(tmp);
         if(tmp > 255)return 0;

         /* network order 3-() */
```

```
          dst[3-(points-n)]= tmp;

          start= point;
          }
          return 1;
     #endif

     default:
          return 0;
     }
     #else
     /* IPv4 format for the level */
     return inet_aton(buf, dst);
     #endif
}
#endif

/* spline: chars per line */
void ip_nat_print_hex(buf, len, spline)
const void *buf;
unsigned len;
unsigned spline;
#if 1
{
unsigned n,col;
u8_t     *p;

     if(!buf)return;
     if(!len)return;
```

```
    if(!spline)return;

    col= 0;
    p= buf;
    for( n=0; n<len; ++n, ++p){
        printf("0x%02x ", *p);

        col+= 1;
        if(col >= spline){
            printf("\n");
            col= 0;
        }
    }
    if(col)printf("\n");
}
#endif

/**/
PUBLIC void ip_nat_init(void)
#if 1
{
    unsigned n;

    /**/
    memset(ip_nat, 0, sizeof(ip_nat));
    ip_nat_items= sizeof(init_ip_nat)/sizeof(init_ip_nat_t);
    if( ip_nat_items > IP_NAT_ITEMS )ip_nat_items=IP_NAT_ITEMS;

    for(n=0; n<ip_nat_items; ++n)
```

```
    {
        mk_init( &ip_nat[n], &init_ip_nat[n] );

        #if DVI_I > 2
        printf("%02u: loc: ", n); writeIpAddr(ip_nat[n].loc);
        printf(" dst: ", n); writeIpAddr(ip_nat[n].dst.ipv4);
        printf("\n");
        #endif
    }

    /**/
    mapped_own_IPv4= 0;
    mk_init_loc( &mapped_own_IPv4, &init_mapped_own_IPv4 );

    /**/
    own_IPv4= 0;
    mk_init_loc( &own_IPv4, &init_own_IPv4 );

    /**/
    memset(&own_IPv4r2, 0, sizeof(own_IPv4r2));
    mk_init_dst( &own_IPv4r2, &init_own_IPv4r2 );

    #if DVI_I > 2
    printf("own IPv4r2: "); writeIpAddr(own_IPv4r2.ipv4);
    printf("\n");
    #endif

    /**/
    IPv4r2_src= 0;
```

```
    mk_init_loc( &IPv4r2_src, &init_IPv4r2_src );

    /**/
    memset(ip_opt_gc, 0, sizeof(ip_opt_gc));
    mk_gc(ip_opt_gc, init_IPv4r2_gate_cmd);

    /**/
    IPv4r2_UDP_dst= 0;
    mk_init_loc( &IPv4r2_UDP_dst, &init_IPv4r2_UDP_dst );

    IPv4r2_UDP_port= htons(init_IPv4r2_UDP_port);

    IPv4r2_UDP_uni_dst= 0;
    mk_init_loc( &IPv4r2_UDP_uni_dst, &init_IPv4r2_UDP_uni_dst );
}
#endif

/* must bf_afree(src_data) if return not the same */
PUBLIC acc_t *ip_nat_write(ip_port, data)
ip_port_t *ip_port;
acc_t *data;
#if 1
{
    /* vars */
    #if 1
    size_t        data_len;
    acc_t         *tmp_pack, *tmp_pack1;
    /* IP header parts: def_hdr, add_src+dst_opt, hdr_opt==prev_body, body */
    acc_t         *prev_body;
```

```
    ip_hdr_t *ip_hdr;
    u32_t         hdr_opt_len;
    u32_t         hdr_ver, hdr_len, ip_size;

    u32_t         add_hdr_opt_tail;
    u8_t      align_opt;

    u32_t         add_src_hdr_opt_len;
    u8_t      add_src_hdr_opt[IP_R2_MTU_CTRL];

    u32_t         add_dst_hdr_opt_len;
    u8_t      add_dst_hdr_opt[IP_R2_MTU_CTRL];

    u8_t      *p;
    u32_t         n;

    ipaddr_t old_ih_dst, old_ih_src;
    char      is_re_chksum;

    tcp_hdr_t*tcp_hdr;
    udp_hdr_t*udp_hdr;

    ip_nat_ps_hdr_tcp_t
                  ps_hdr_tcp;
    ip_nat_ps_hdr_udp_t
                  ps_hdr_udp;

    ip_hdr_t *ip_hdr_udp;
```

```
    u8_t      *udp_hdr_sign;
    u32_t          ip_size_udp;
    char      is_IPv4r2_opt, is_IPv4r2_do_UDP;
    #endif

    /* check src is own IPv4 of NAT */
    if( is_own_IPv4 ){
         if( ip_port->ip_ipaddr != own_IPv4 ){
              return data;
         }
    }

    data_len= bf_bufsize(data);
    #if DVI_W > 1
    printf( "write outcomimg bs: %u\n", data_len );
    #endif

    /* check IP hdr exist */
    #if 1
    if( data_len < IP_MIN_HDR_SIZE ){
         #if DVI_W > 1
         printf("write: no ip header\n");
         #endif

         bf_afree(data);
         return 0;
    }
    #endif
```

```
    /* get IP hdr params */
    #if 1
    /* assume data already at least IP_MIN_HDR_SIZE continuous memory */
    /* but ensure we have continuous memory access to ip header */
    data= bf_packIffLess(data, IP_MIN_HDR_SIZE);

    ip_hdr= (ip_hdr_t *)ptr2acc_data(data);
    hdr_ver= (ip_hdr->ih_vers_ihl >> 4) & IH_IHL_MASK;
    hdr_len= (ip_hdr->ih_vers_ihl & IH_IHL_MASK)*4;
    ip_size= ntohs(ip_hdr->ih_length);

    old_ih_src= ip_hdr->ih_src;
    old_ih_dst= ip_hdr->ih_dst;
    is_re_chksum= 0;
    is_IPv4r2_opt= 0;
    is_IPv4r2_do_UDP= 0;
    #endif

    /* check map own IPv4 to IPv4r2 NAT local address area */
    #if 1
    if(
        is_map_own_IPv4
        && ( ip_hdr->ih_dst == mapped_own_IPv4 )
    ){
        /* swap src and dst addresses */
        ip_hdr->ih_dst = old_ih_src;
        ip_hdr->ih_src = old_ih_dst;

        return data;
```

```
    }
    #endif

    /* temporary check loopback here to do not change ip_write */
    #if 1
    p= (u8_t *)&ip_hdr->ih_dst;
    if(
         ( ip_hdr->ih_dst == ip_port->ip_ipaddr )
         ||( p[0] == 127 )
    ){
         return data;
    }
    #endif

    /* check IP frame is correct */
    #if 1
    if(
         ( hdr_ver != 4 )
         ||( hdr_len > ip_size )
         ||( ip_size > data_len)
    ){
         #if DVI_W > 1
         printf("read: wrong ip frame: "
                "hdr_ver: %u, hdr_len: %u, ip_size: %u\n",
                hdr_ver, hdr_len, ip_size);
         #endif

         bf_afree(data);
         return 0;
```

```
    }
    #endif

    /* split old ip pack to be ready to add new ip header options */
    #if 1
    tmp_pack= bf_cut(data, 0, IP_MIN_HDR_SIZE);
    /*
    ensure we have continuous memory access to ip header
    and empty tmp_pack->acc_next
    */
    tmp_pack= bf_packIffLess(tmp_pack, IP_MIN_HDR_SIZE);
    prev_body= tmp_pack;
    tmp_pack1 = tmp_pack;

    ip_hdr= (ip_hdr_t *)ptr2acc_data(tmp_pack);
    #endif

    /* split old ip header options */
    #if 1
    hdr_opt_len= hdr_len - IP_MIN_HDR_SIZE;
    if(hdr_opt_len)
    {
        tmp_pack= bf_cut(data, IP_MIN_HDR_SIZE, hdr_opt_len);
        /*
        ensure we have continuous memory access to ip header options
        and empty tmp_pack->acc_next
        */
        tmp_pack= bf_packIffLess(tmp_pack, hdr_opt_len);
        assert(!prev_body->acc_next);
```

```
          prev_body->acc_next= tmp_pack;
          prev_body= tmp_pack;
     }
     #endif

     /* split ip pack body */
     #if 1
     tmp_pack= bf_delhead(data, hdr_len);
     #if DVI_W > 1
     printf( "write body: %u, hdr: %u, ip_size: %u\n", bf_bufsize(tmp_pack), hdr_len,
ip_size );
     #endif

     assert(!prev_body->acc_next);
     prev_body->acc_next= tmp_pack;
     data= tmp_pack1;
     #if DVI_W > 1
     printf( "write repacked: %u\n", bf_bufsize(data) );
     #endif
     #endif

     /* check src NAT */
     add_src_hdr_opt_len= 0;
     for(;is_own_IPv4r2;){
     #if 1
     /*
     else assume ih_src is already correct if requested dst NAT
     ip_hdr->ih_src == ip_nat_wan_ip;
     */
```

```
/* src NAT for specified IPv4 */
if( is_IPv4r2_src ){
     if( IPv4r2_src != old_ih_src )break;
}else{
     /* src NAT for local IPv4 */
     if( ip_port->ip_ipaddr != old_ih_src )break;
}

/* do src NAT */
is_re_chksum= 1;
is_IPv4r2_do_UDP= 1;

/* src IPv4 */
ip_hdr->ih_src = own_IPv4r2.ipv4;

/* src IPv4r2 options */
#if 1
if(own_IPv4r2.ip_opt_flags & IP_NAT_R2_EN_B45){
     memcpy(
          add_src_hdr_opt + add_src_hdr_opt_len,
          own_IPv4r2.b.ip_opt_b45,
          8);
     /* set IPv4r2 src addr extention mark */
     add_src_hdr_opt[add_src_hdr_opt_len+2] &= 0x1f;
     add_src_hdr_opt[add_src_hdr_opt_len+2] |= 0x40;
     add_src_hdr_opt_len+= 8;
}
```

```
    if(own_IPv4r2.ip_opt_flags & IP_NAT_R2_EN_U12){
         memcpy(
              add_src_hdr_opt + add_src_hdr_opt_len,
              own_IPv4r2.u.ip_opt_u12,
              4);
         /* set IPv4r2 src addr extention mark */
         add_src_hdr_opt[add_src_hdr_opt_len+2] &= 0xef;
         add_src_hdr_opt[add_src_hdr_opt_len+2] |= 0x00;
         add_src_hdr_opt_len+= 4;
    }

    if(own_IPv4r2.ip_opt_flags & IP_NAT_R2_EN_V6){
         memcpy(
              add_src_hdr_opt + add_src_hdr_opt_len,
              own_IPv4r2.v.ip_opt_v6,
              20);
         /* set IPv4r2 src addr extention mark */
         add_src_hdr_opt[add_src_hdr_opt_len+2] &= 0x00;
         add_src_hdr_opt[add_src_hdr_opt_len+3] &= 0x00;
         add_src_hdr_opt[add_src_hdr_opt_len+2] |= 0x00;
         add_src_hdr_opt_len+= 20;
    }
    #endif

    break;
    /*for(;is_own_IPv4r2;)*/
    #endif
    }
    #if DVI_W > 1
```

```
    printf( "write src opts: %u\n", add_src_hdr_opt_len );
    ip_nat_print_hex(add_src_hdr_opt, add_src_hdr_opt_len, 16);
    #endif

    /* check dst NAT */
    add_dst_hdr_opt_len= 0;
    for(n=0; n<ip_nat_items; ++n){
    #if 1
    if( ip_nat[n].loc != ip_hdr->ih_dst )continue;

    /* do dst NAT */
    is_re_chksum= 1;
    is_IPv4r2_do_UDP= 1;

    /* dst IPv4 */
    ip_hdr->ih_dst = ip_nat[n].dst.ipv4;

    /* dst IPv4r2 options */
    #if 1
    if(ip_nat[n].dst.ip_opt_flags & IP_NAT_R2_EN_B45){
         memcpy(
              add_dst_hdr_opt + add_dst_hdr_opt_len,
              ip_nat[n].dst.b.ip_opt_b45,
              8);
         /* set IPv4r2 dst addr extention mark */
         add_dst_hdr_opt[add_dst_hdr_opt_len+2] &= 0x1f;
         add_dst_hdr_opt[add_dst_hdr_opt_len+2] |= 0x80;
         add_dst_hdr_opt_len+= 8;
    }
```

```
    if(ip_nat[n].dst.ip_opt_flags & IP_NAT_R2_EN_U12){
         memcpy(
              add_dst_hdr_opt + add_dst_hdr_opt_len,
              ip_nat[n].dst.u.ip_opt_u12,
              4);
         /* set IPv4r2 dst addr extention mark */
         add_dst_hdr_opt[add_dst_hdr_opt_len+2] &= 0xef;
         add_dst_hdr_opt[add_dst_hdr_opt_len+2] |= 0x10;
         add_dst_hdr_opt_len+= 4;
    }

    if(ip_nat[n].dst.ip_opt_flags & IP_NAT_R2_EN_V6){
         memcpy(
              add_dst_hdr_opt + add_dst_hdr_opt_len,
              ip_nat[n].dst.v.ip_opt_v6,
              20);
         /* set IPv4r2 dst addr extention mark */
         add_dst_hdr_opt[add_dst_hdr_opt_len+2] &= 0x00;
         add_dst_hdr_opt[add_dst_hdr_opt_len+3] &= 0x00;
         add_dst_hdr_opt[add_dst_hdr_opt_len+2] |= 0x80;
         add_dst_hdr_opt_len+= 20;
    }
    #endif

    break;
    /*for n*/
    #endif
    }
```

```
#if DVI_W > 1
printf( "write dst opts: %u\n", add_dst_hdr_opt_len );
ip_nat_print_hex(add_dst_hdr_opt, add_dst_hdr_opt_len, 16);
#endif

/* add gate cmd if IPv4 src to IPv4r2 dst */
/*
in IPv4r2.1 mode IPv4r2 software avoids the IPv4 NAT
itself emits IPv4r2 dst and does not use IPv4 servers
and itself generates own gate_cmd=1 if needed

if IPv4 NAT in process (add_dst_hdr_opt_len exist)
that means IPv4r2.0 mode in use
and IPv4 dst was generated by IPv4 software
*/
if(
     !add_src_hdr_opt_len
     && add_dst_hdr_opt_len
     && is_IPv4r2_gate_cmd
){
#if 1
     /**/
     memcpy(
          add_src_hdr_opt + add_src_hdr_opt_len,
          ip_opt_gc,
          ip_opt_gc[1]
          );
     add_src_hdr_opt_len += ip_opt_gc[1];
```

```
        /* align by 0x01 to 4 boundary for debug purpose */
        #if 1
        add_src_hdr_opt[add_src_hdr_opt_len]= 0x01;
        add_src_hdr_opt_len += 1;
        #endif

        #if DVI_W > 1
        printf( "write src+gc opts: %u\n", add_src_hdr_opt_len );
        ip_nat_print_hex(add_src_hdr_opt, add_src_hdr_opt_len, 16);
        #endif
    #endif
    }

    /* join src+dst if dst exist */
    if(add_dst_hdr_opt_len){
    #if 1
        /**/
        memcpy(
            add_src_hdr_opt + add_src_hdr_opt_len,
            add_dst_hdr_opt,
            add_dst_hdr_opt_len
            );
        add_src_hdr_opt_len += add_dst_hdr_opt_len;
    #endif
    }
    #if DVI_W > 1
    printf( "write src+dst opts: %u\n", add_src_hdr_opt_len );
    ip_nat_print_hex(add_src_hdr_opt, add_src_hdr_opt_len, 16);
    #endif
```

```
/* add IPv4r2 options if exist */
if (add_src_hdr_opt_len){
#if 1

     /*
     if there are old options align new to 4 by option 0x01
     else align new to 4 by option 0x00
     */
     align_opt = (hdr_opt_len)? 0x01: 0x00;

     /* ?new IPv4r2 are not aligned to 4 */
     add_hdr_opt_tail = add_src_hdr_opt_len % 4;
     if(add_hdr_opt_tail){
          memset(
               add_src_hdr_opt + add_src_hdr_opt_len,
               align_opt,
               4-add_hdr_opt_tail
               );
          add_src_hdr_opt_len += 4-add_hdr_opt_tail;
     }

     /* ckeck header size is less or equal IP_MAX_HDR_SIZE */
     hdr_len += add_src_hdr_opt_len;
     if( hdr_len > IP_MAX_HDR_SIZE ){
          bf_afree(data);
          return 0;
     }
```

```
        ip_size += add_src_hdr_opt_len;
        if( ip_size > IP_MAX_PACKSIZE ){
             bf_afree(data);
             return 0;
        }

        /* add new options */
        tmp_pack= bf_memreq(add_src_hdr_opt_len);
        /* ensure we have continuous memory access to */
        tmp_pack= bf_packIffLess(tmp_pack, add_src_hdr_opt_len);
        assert(!tmp_pack->acc_next);
        tmp_pack->acc_next= data->acc_next;
        data->acc_next= tmp_pack;

        /* move body pointer if there were no options */
        if(!hdr_opt_len)prev_body= tmp_pack;

        /**/
        memcpy(
             ptr2acc_data(tmp_pack),
             add_src_hdr_opt,
             add_src_hdr_opt_len
             );

        /* do correct IPv4 header */
        ip_hdr->ih_vers_ihl&= ~IH_IHL_MASK;
        ip_hdr->ih_vers_ihl|= (hdr_len/4) & IH_IHL_MASK;
        ip_hdr->ih_length= htons(ip_size);
```

```
        is_IPv4r2_opt= 1;

    #endif
    }

    /* src+dst IPv4 recount chksum */
    if(is_re_chksum){
    #if 1
    /* TCP/UDP chksum */
    switch(ip_hdr->ih_proto){
    #if 1

        /* TCP chksum */
        #if 1
        case 0x06:
        /* ensure we have continuous memory access to TCP header */
        prev_body->acc_next = bf_packIffLess(prev_body->acc_next, TCP_MIN_HDR_SIZE);

        tcp_hdr = (tcp_hdr_t*)ptr2acc_data(prev_body->acc_next);

        memset( &ps_hdr_tcp, 0, sizeof(ps_hdr_tcp) );
        ps_hdr_tcp.not_old_ih_src= ~old_ih_src;
        ps_hdr_tcp.not_old_ih_dst= ~old_ih_dst;
        ps_hdr_tcp.new_ih_src= ip_hdr->ih_src;
        ps_hdr_tcp.new_ih_dst= ip_hdr->ih_dst;

        tcp_hdr->th_chksum = ~oneC_sum(~(tcp_hdr->th_chksum), &ps_hdr_tcp,
sizeof(ps_hdr_tcp));
        break;
```

```
        #endif

        /* UDP chksum */
        #if 1
        case 0x11:
        /* ensure we have continuous memory access to UDP header */
        prev_body->acc_next= bf_packIffLess(prev_body->acc_next, UDP_HDR_SIZE);

        udp_hdr = (udp_hdr_t*)ptr2acc_data(prev_body->acc_next);

        #if 1
        udp_hdr->uh_chksum = 0;

        #else
        if(udp_hdr->uh_chksum){

        memset( &ps_hdr_udp, 0, sizeof(ps_hdr_udp) );
        ps_hdr_udp.not_old_ih_src= ~old_ih_src;
        ps_hdr_udp.not_old_ih_dst= ~old_ih_dst;
        ps_hdr_udp.new_ih_src= ip_hdr->ih_src;
        ps_hdr_udp.new_ih_dst= ip_hdr->ih_dst;

        udp_hdr->uh_chksum = ~oneC_sum(~(udp_hdr->uh_chksum), &ps_hdr_udp,
sizeof(ps_hdr_udp));
        }
        #endif
        break;
        #endif
```

```
    /*switch(ip_hdr->ih_proto)*/
    #endif
    }
    #endif
    }

    /* IPv4 chksum is mandatory for write */
    #if 1
    /* IPv4 header buffers must be joined for ip_write */
    tmp_pack= prev_body->acc_next;
    prev_body->acc_next= 0;
    #if DVI_W > 1
    printf( "write hdr bs: %u, hdr_len: %u\n", bf_bufsize(data), hdr_len );
    #endif
    assert( bf_bufsize(data) >= hdr_len );

    data = bf_pack(data);
    assert(!data->acc_next);
    data->acc_next= tmp_pack;
    #if DVI_W > 1
    printf( "write pack bs: %u, ip_size: %u\n", bf_bufsize(data), ip_size );
    #endif
    assert( bf_bufsize(data) >= ip_size );

    ip_hdr= (ip_hdr_t *)ptr2acc_data(data);
    ip_hdr_chksum(ip_hdr, hdr_len);
    #endif

    #if DVI_W > 1
```

```
    ip_nat_print_hex(ip_hdr, hdr_len, 16);
    #endif

    /* pack to IPv4 UDP for IPv4r2 */
    for(;
         is_IPv4r2_UDP
         && (is_IPv4r2_opt || is_IPv4r2_do_UDP)
         ;
    ){
    #if 1

    /* check min UI frame body */
    #if 1
    if( ip_size < IP_MIN_HDR_SIZE ){
         #if DVI_W > 1
         printf( "write can not make UDP frame: "
                 "no ip body: ip_size: %u\n", ip_size );
         ip_nat_print_hex(ptr2acc_data(data), ip_size, 16);
         #endif

         break;
    }
    #endif

    /**/
    ip_hdr_udp= (ip_hdr_t *)add_dst_hdr_opt;
    udp_hdr= (udp_hdr_t *)&add_dst_hdr_opt[IP_MIN_HDR_SIZE];
    udp_hdr_sign= &add_dst_hdr_opt[IP_MIN_HDR_SIZE + UDP_HDR_SIZE];
```

```
    add_dst_hdr_opt_len= IP_MIN_HDR_SIZE + UDP_HDR_SIZE + sizeof(IPv4r2_UDP_sign);
    ip_size_udp= ip_size + add_dst_hdr_opt_len;

    /* do IPv4 UDP frame: IPv4 header */
    #if 1
    ip_hdr_udp->ih_vers_ihl= 0x40;
    ip_hdr_udp->ih_vers_ihl|= (IP_MIN_HDR_SIZE/4) & IH_IHL_MASK;
    ip_hdr_udp->ih_length= htons(ip_size_udp);

    ip_hdr_udp->ih_tos= ip_hdr->ih_tos;
    ip_hdr_udp->ih_id= 0;
    ip_hdr_udp->ih_flags_fragoff= htons(IH_DONT_FRAG);
    ip_hdr_udp->ih_ttl= ip_hdr->ih_ttl;
    ip_hdr_udp->ih_proto= IPPROTO_UDP;

    ip_hdr_udp->ih_src= is_IPv4r2_UDP_own_src? ip_port->ip_ipaddr: ip_hdr->ih_src;
    ip_hdr_udp->ih_dst= is_IPv4r2_UDP_dst? IPv4r2_UDP_dst: ip_hdr->ih_dst;

    ip_hdr_chksum(ip_hdr_udp, IP_MIN_HDR_SIZE);
    #endif

    /* do IPv4 UDP frame: UDP header */
    #if 1
    udp_hdr->uh_length= htons(ip_size_udp - IP_MIN_HDR_SIZE);
    udp_hdr->uh_src_port= IPv4r2_UDP_port;
    udp_hdr->uh_dst_port= IPv4r2_UDP_port;
    udp_hdr->uh_chksum= 0;
    #endif
```

```
    /* fill IPv4 UDP frame: UDP sign */
    memcpy(
         udp_hdr_sign,
         IPv4r2_UDP_sign,
         sizeof(IPv4r2_UDP_sign)
         );

    /* add UDP frame to data */
    tmp_pack= bf_memreq(add_dst_hdr_opt_len);
    /* ensure we have continuous memory access to */
    tmp_pack= bf_packIffLess(tmp_pack, add_dst_hdr_opt_len);
    assert(!tmp_pack->acc_next);
    tmp_pack->acc_next= data;
    data= tmp_pack;

    memcpy(
         ptr2acc_data(tmp_pack),
         add_dst_hdr_opt,
         add_dst_hdr_opt_len
         );

    #if DVI_W > 1
    printf( "write UDP frame add_size: %u, total ip_size: %u\n",
              add_dst_hdr_opt_len, bf_bufsize(data) );
    ip_nat_print_hex(ptr2acc_data(data), add_dst_hdr_opt_len, 16);
    #endif

    break;
    #endif
```

```
    /*for(is_IPv4r2_UDP)*/
    }

    /**/
    return data;
}
#endif

/*walk over IPv4 options*/
/*
exmple of network performance && source code structure optimization:
    single time IPv4 options parse
    return dst tmp_hdr_opt buf size with IPv4r2 options removed
    (return tmp_hdr_opt_len)

to more network performance (from IPv4 up to IPv6 equ network):
for example IPv4r2.0
first step: check IPv4r2.x options at fixed offsets
    size= 0;
    hdr_opt[20+size] B45 src, then U12 src, then v6 src
    then B45 dst, then U12 dst, then v6 dst
    then IPv4r2 addr extention options limiter

if failed second step:
    parse options in general way
*/
PUBLIC unsigned ip_nat_walk_opt(
                /*dst IPv4 options (IPv4r2 options removed)*/
                u8_t *tmp_hdr_opt,
```

```
                    /*src IPv4 options + IPv4r2 options*/
                    u8_t *hdr_opt,
                    u32_t hdr_opt_len,
                    /*found and stored IPv4r2 options*/
                    ip_nat_dst_t *src_IPv4r2,
                    ip_nat_dst_t *dst_IPv4r2
                    )
#if 1
{
unsigned i, tmp_hdr_opt_len;

char     is_addr_found;

u8_t     *hdr_src, *hdr_dst;
ip_nat_dst_t *ptr_IPv4r2;

    /* init ip_nat_dst_t(s) */
    #if 1
    if(src_IPv4r2)src_IPv4r2->ip_opt_flags= 0;
    if(dst_IPv4r2)dst_IPv4r2->ip_opt_flags= 0;
    #endif

    /*check input params*/
    #if 1
    if( !hdr_opt || !hdr_opt_len )return 0;
    /* zero dst is allowed with intention */
    #endif

    /* init for */
```

```
    #if 1
    hdr_src= hdr_opt;
    hdr_dst= tmp_hdr_opt;
    tmp_hdr_opt_len= 0;
    #endif

    /*walk over IPv4 options*/
    /*
    assume for options alignment free placement
    if we do full parse on read, then can
         remove all extra 0, 1 space options
    if we do not full parse on write, then can add
         1 to align 4 of intermediate options chunk
         0 to align 4 of last options chunk
    */
    for( i=0; i<hdr_opt_len; ){
    #if 1
         /* single byte options opcodes */
         #if 1
         if( *hdr_src == 0 ){ break; }

         if( *hdr_src == 1 ){
              /* skip leading align if(tmp_hdr_opt_len) */
              /* skip leading align
              if(hdr_dst){ *hdr_dst++ = 1; }
              tmp_hdr_opt_len+= 1;
              */

              i+= 1;
```

```
            /*must be last*/
            hdr_src+= 1;
            continue;
        }
        #endif

        /* invalid options */
        if( (i+2) > hdr_opt_len )break;
        if( (i+hdr_src[1]) > hdr_opt_len )break;
        if( hdr_src[1] < 2 )break;

        /* two byte options opcodes */
        switch(hdr_src[0]){

            case IP_OPT_R2_1:
            /*B45*/
            #if 1
            if( hdr_src[1] == 8 ){

            /* check IPv4r2 opcode */

            /* check IPv4r2 src/dst sign */
            #if  1
            is_addr_found= 0;
                /* check IPv4r2 src addr extention mark */
            if( (hdr_src[2] & 0xe0) == 0x40 ){
                ptr_IPv4r2= src_IPv4r2;
                is_addr_found= 1;
            }
```

```
            /* check IPv4r2 dst addr extention mark */
        else if( (hdr_src[2] & 0xe0) == 0x80 ){
            ptr_IPv4r2= dst_IPv4r2;
            is_addr_found= 1;
        }
        if(!is_addr_found)break;
        #endif

        /* found B45, copy to ptr_IPv4r2 */
        /* and skip invalid repeated B45 */
        #if 1
        if(ptr_IPv4r2){
        if( !(ptr_IPv4r2->ip_opt_flags & IP_NAT_R2_EN_B45) ){

        ptr_IPv4r2->ip_opt_flags|= IP_NAT_R2_EN_B45;
        memcpy(
            ptr_IPv4r2->b.ip_opt_b45,
            hdr_src,
            hdr_src[1]
            );
        }}
        #endif

        /* remove from dst IP options*/
        i+= hdr_src[1];
        /*must be last*/
        hdr_src+= hdr_src[1];
        continue;
        }
```

```
#endif

/*V6*/
#if 1
if( hdr_src[1] == 20 ){

/* check IPv4r2 opcode */
if( (hdr_src[2] & 0x7f) != 0x00 )break;
if( (hdr_src[3] & 0xff) != 0x00 )break;

/* check IPv4r2 src/dst sign */
#if 1
is_addr_found= 0;
     /* check IPv4r2 src addr extention mark */
if( (hdr_src[2] & 0x80) == 0x00 ){
     ptr_IPv4r2= src_IPv4r2;
     is_addr_found= 1;
}
     /* check IPv4r2 dst addr extention mark */
else if( (hdr_src[2] & 0x80) == 0x80 ){
     ptr_IPv4r2= dst_IPv4r2;
     is_addr_found= 1;
}
if(!is_addr_found)break;
#endif

/* found V6, copy to ptr_IPv4r2 */
/* and skip invalid repeated V6 */
#if 1
```

```
            if(ptr_IPv4r2){
            if( !(ptr_IPv4r2->ip_opt_flags & IP_NAT_R2_EN_V6) ){

            ptr_IPv4r2->ip_opt_flags|= IP_NAT_R2_EN_V6;
            memcpy(
                 ptr_IPv4r2->v.ip_opt_v6,
                 hdr_src,
                 hdr_src[1]
                 );
            }}
            #endif

            /* remove from dst IP options*/
            i+= hdr_src[1];
            /*must be last*/
            hdr_src+= hdr_src[1];
            continue;
            }
            #endif
            break;

            case IP_OPT_R2_3:
            /*GC*/
            #if 1
            if( hdr_src[1] == 3 ){

            /* check IPv4r2 opcode */
            if( (hdr_src[2] & 0xf0) != 0x10 )break;
```

```
/* remove gate command got for non gate host */

/* remove from dst IP options*/
i+= hdr_src[1];
/*must be last*/
hdr_src+= hdr_src[1];
continue;
}
#endif

/*U12*/
#if 1
if( hdr_src[1] == 4 ){

/* check IPv4r2 opcode */
if( (hdr_src[2] & 0xe0) != 0x20 )break;

/* check IPv4r2 src/dst sign */
#if 1
is_addr_found= 0;
     /* check IPv4r2 src addr extention mark */
if( (hdr_src[2] & 0x10) == 0x00 ){
     ptr_IPv4r2= src_IPv4r2;
     is_addr_found= 1;
}
     /* check IPv4r2 dst addr extention mark */
else if( (hdr_src[2] & 0x10) == 0x10 ){
     ptr_IPv4r2= dst_IPv4r2;
     is_addr_found= 1;
```

```
            }
            if(!is_addr_found)break;
            #endif

            /* found U12, copy to ptr_IPv4r2 */
            /* and skip invalid repeated U12 */
            #if 1
            if(ptr_IPv4r2){
            if( !(ptr_IPv4r2->ip_opt_flags & IP_NAT_R2_EN_U12) ){

            ptr_IPv4r2->ip_opt_flags|= IP_NAT_R2_EN_U12;
            memcpy(
                 ptr_IPv4r2->u.ip_opt_u12,
                 hdr_src,
                 hdr_src[1]
                 );
            }}
            #endif

            /* remove from dst IP options*/
            i+= hdr_src[1];
            /*must be last*/
            hdr_src+= hdr_src[1];
            continue;
            }
            #endif
            break;
        }
```

```
          /*default: copy IP option to dst*/
          #if 1
          tmp_hdr_opt_len+= hdr_src[1];
          i+= hdr_src[1];

          if(hdr_dst){
               memcpy(
                    hdr_dst,
                    hdr_src,
                    hdr_src[1]
                    );
               hdr_dst+= hdr_src[1];
          }
          /*must be last*/
          hdr_src+= hdr_src[1];
          continue;
          #endif

     /*for(;i<hdr_opt_len;)*/
     #endif
     }

     return tmp_hdr_opt_len;
}
#endif

/**/
PUBLIC char ip_nat_cmp_B45(
                    ip_nat_dst_t *parsed_IPv4r2,
```

```
                    ip_nat_dst_t *req_IPv4r2
                    )
#if 1
{
     if( !parsed_IPv4r2 || !req_IPv4r2 )return 0;

     /*exist parsed B45*/
     if( parsed_IPv4r2->ip_opt_flags & IP_NAT_R2_EN_B45 ){
     if( !(req_IPv4r2->ip_opt_flags & IP_NAT_R2_EN_B45) )return 0;

     if(
          (parsed_IPv4r2->b.ip_opt_b45[2] & 0x1f)
          != (req_IPv4r2->b.ip_opt_b45[2] & 0x1f)
          )return 0;

     if( memcmp(
               &parsed_IPv4r2->b.ip_opt_b45[3],
               &req_IPv4r2->b.ip_opt_b45[3],
               req_IPv4r2->b.ip_opt_b45[1]-3
               )
          )return 0;

     return 1;
     }

     /* not exist parsed B45*/
     if( req_IPv4r2->ip_opt_flags & IP_NAT_R2_EN_B45 )return 0;
     return 1;
}
```

```
#endif

/**/
PUBLIC char ip_nat_cmp_U12(
                  ip_nat_dst_t *parsed_IPv4r2,
                  ip_nat_dst_t *req_IPv4r2
                  )
#if 1
{
     if( !parsed_IPv4r2 || !req_IPv4r2 )return 0;

     /*exist parsed U12*/
     if( parsed_IPv4r2->ip_opt_flags & IP_NAT_R2_EN_U12 ){
     if( !(req_IPv4r2->ip_opt_flags & IP_NAT_R2_EN_U12) )return 0;

     if(
          (parsed_IPv4r2->u.ip_opt_u12[2] & 0x0f)
          != (req_IPv4r2->u.ip_opt_u12[2] & 0x0f)
          )return 0;

     if( memcmp(
               &parsed_IPv4r2->u.ip_opt_u12[3],
               &req_IPv4r2->u.ip_opt_u12[3],
               req_IPv4r2->u.ip_opt_u12[1]-3
               )
          )return 0;

     return 1;
     }
```

```
    /* not exist parsed U12*/
    if( req_IPv4r2->ip_opt_flags & IP_NAT_R2_EN_U12 )return 0;
    return 1;
}
#endif

/**/
PUBLIC char ip_nat_cmp_V6(
                ip_nat_dst_t *parsed_IPv4r2,
                ip_nat_dst_t *req_IPv4r2
                )
#if 1
{
    if( !parsed_IPv4r2 || !req_IPv4r2 )return 0;

    /*exist parsed V6*/
    if( parsed_IPv4r2->ip_opt_flags & IP_NAT_R2_EN_V6 ){
    if( !(req_IPv4r2->ip_opt_flags & IP_NAT_R2_EN_V6) )return 0;

    if( memcmp(
            &parsed_IPv4r2->v.ip_opt_v6[4],
            &req_IPv4r2->v.ip_opt_v6[4],
            req_IPv4r2->v.ip_opt_v6[1]-4
            )
        )return 0;

    return 1;
    }
```

```
    /* not exist parsed V6*/
    if( req_IPv4r2->ip_opt_flags & IP_NAT_R2_EN_V6 )return 0;
    return 1;
}
#endif

/*return hdr_opt_len*/
PUBLIC void ip_nat_buf_opt(
                  /*to hdr_opt buf[hdr_opt_len]*/
                  u8_t *hdr_opt,
                  u32_t *hdr_opt_len,
                  /*from parsed_IPv4r2 struct*/
                  ip_nat_dst_t *parsed_IPv4r2
                  )
#if 1
{
    if( !hdr_opt || !hdr_opt_len || !parsed_IPv4r2 )return;

    if( parsed_IPv4r2->ip_opt_flags & IP_NAT_R2_EN_B45 ){
    memcpy(
         hdr_opt + *hdr_opt_len,
         parsed_IPv4r2->b.ip_opt_b45,
         parsed_IPv4r2->b.ip_opt_b45[1]
         );
    *hdr_opt_len+= parsed_IPv4r2->b.ip_opt_b45[1];
    }

    if( parsed_IPv4r2->ip_opt_flags & IP_NAT_R2_EN_U12 ){
```

```
     memcpy(
          hdr_opt + *hdr_opt_len,
          parsed_IPv4r2->u.ip_opt_u12,
          parsed_IPv4r2->u.ip_opt_u12[1]
          );
     *hdr_opt_len+= parsed_IPv4r2->u.ip_opt_u12[1];
     }

     if( parsed_IPv4r2->ip_opt_flags & IP_NAT_R2_EN_V6 ){
     memcpy(
          hdr_opt + *hdr_opt_len,
          parsed_IPv4r2->v.ip_opt_v6,
          parsed_IPv4r2->v.ip_opt_v6[1]
          );
     *hdr_opt_len+= parsed_IPv4r2->v.ip_opt_v6[1];
     }

}
#endif

/**/
PUBLIC acc_t *ip_nat_read(ip_port, data)
ip_port_t *ip_port;
acc_t *data;
#if 1
{
     /*vars*/
     #if 1
     size_t        data_len;
```

```
    acc_t          *tmp_pack, *tmp_pack1;
    /* IP header parts: def_hdr, hdr_opt==prev_body, body */
    acc_t          *prev_body;

    ip_hdr_t  *ip_hdr;
    u8_t      *hdr_opt;
    u32_t          hdr_opt_len;
    u32_t          hdr_ver, hdr_len, ip_size;

    u32_t          hdr_opt_tail;
    u8_t      align_opt;

    char           is_opt_parsed;
    ip_nat_dst_t   src_IPv4r2, dst_IPv4r2;
    u8_t           tmp_hdr_opt[IP_R2_MTU_CTRL];
    u32_t              tmp_hdr_opt_len;
    u8_t           restored_hdr_opt[IP_R2_MTU_CTRL];
    u32_t              restored_hdr_opt_len;

    char      is_hdr_opt_found;
    char      is_hdr_opt_found_B45;
    char      is_hdr_opt_found_U12;
    char      is_hdr_opt_found_V6;

    char      is_src_nat_done;
    char      is_dst_nat_done;

    u32_t          n;
```

```
    u8_t      *old_hdr_opt;
    u32_t          old_hdr_opt_len;
    ipaddr_t  old_ih_dst, old_ih_src;
    char      is_re_chksum;

    tcp_hdr_t*tcp_hdr;
    udp_hdr_t*udp_hdr;

    ip_nat_ps_hdr_tcp_t
                   ps_hdr_tcp;
    ip_nat_ps_hdr_udp_t
                   ps_hdr_udp;

    u8_t      *udp_hdr_sign;
    #endif

    /* check src is own IPv4 of NAT */
    if( is_own_IPv4 ){
         if( ip_port->ip_ipaddr != own_IPv4 ){
              return data;
         }
    }

    /* unpack from IPv4 UDP for IPv4r2 */
    for(;is_IPv4r2_UDP;){
    #if 1

    data_len = bf_bufsize(data);
    #if DVI_R > 1
```

```
    printf( "read UDP incomimg bs: %u\n", data_len );
    #endif

    /* check IP hdr exist */
    #if 1
    if( data_len < IP_MIN_HDR_SIZE ){
        #if DVI_R > 1
        printf("read: no ip header\n");
        #endif

        bf_afree(data);
        return 0;
    }
    #endif

    /* split old ip pack to be ready to remove old ip header options */
    tmp_pack= bf_cut(data, 0, IP_MIN_HDR_SIZE);
    /* ensure we have continuous memory access to ip header */
    tmp_pack= bf_packIffLess(tmp_pack, IP_MIN_HDR_SIZE);

    ip_hdr= (ip_hdr_t *)ptr2acc_data(tmp_pack);
    hdr_ver= (ip_hdr->ih_vers_ihl >> 4) & IH_IHL_MASK;
    hdr_len= (ip_hdr->ih_vers_ihl & IH_IHL_MASK)*4;
    ip_size= ntohs(ip_hdr->ih_length);

    /* check IP frame is correct */
    #if 1
    if(
        ( hdr_ver != 4 )
```

```
        ||( hdr_len > ip_size )
        ||( ip_size > data_len)
    ){
        #if DVI_R > 1
        printf("read: wrong ip frame: "
                "hdr_ver: %u, hdr_len: %u, ip_size: %u\n",
                hdr_ver, hdr_len, ip_size);
        #endif

        bf_afree(tmp_pack);
        bf_afree(data);
        return 0;
    }
    #endif

    /* check 1 IP frame is IPv4r2 UI */
    #if 1
    if(
        ( !is_IPv4r2_UDP_read_all )
    &&  ( ip_hdr->ih_dst != ip_port->ip_ipaddr )
    &&  ( !is_IPv4r2_UDP_uni_dst
        ||( ip_hdr->ih_dst != IPv4r2_UDP_uni_dst ) )
    ){
        #if DVI_R > 1
        printf( "read UDP: drop is not IPv4r2_UDP UI: %s", "other dst: " );
        writeIpAddr(ip_hdr->ih_dst);
        printf( "\n" );
        #endif
```

```
        bf_afree(tmp_pack);
        break;
    }

    if( ip_hdr->ih_proto != IPPROTO_UDP ){
        #if DVI_R > 1
        printf( "read UDP: drop is not IPv4r2_UDP UI: %s\n", "protocol" );
        #endif

        bf_afree(tmp_pack);
        break;
    }

    /* min UI frame */
    tmp_hdr_opt_len= hdr_len
        + UDP_HDR_SIZE + sizeof(IPv4r2_UDP_sign) + IP_MIN_HDR_SIZE;

    if( ip_size < tmp_hdr_opt_len ){
        #if DVI_R > 1
        printf( "read UDP: drop is not IPv4r2_UDP UI: %s\n", "ip_size" );
        #endif

        bf_afree(tmp_pack);
        break;
    }
    #endif

    tmp_hdr_opt_len= UDP_HDR_SIZE + sizeof(IPv4r2_UDP_sign);
```

```
    /* split UI body */
    tmp_pack1= bf_cut(data, hdr_len, tmp_hdr_opt_len);
    /* ensure we have continuous memory access to ip header */
    tmp_pack1= bf_packIffLess(tmp_pack1, IP_MIN_HDR_SIZE);

    udp_hdr= (udp_hdr_t *)ptr2acc_data(tmp_pack1);
    udp_hdr_sign= ((u8_t *)ptr2acc_data(tmp_pack1)) + UDP_HDR_SIZE;

    /* check 2 IP frame is IPv4r2 UI */
    #if 1
    if( memcmp(
              udp_hdr_sign,
              IPv4r2_UDP_sign,
              sizeof(IPv4r2_UDP_sign)
              )
    ){
         #if DVI_R > 1
         printf( "read UDP: drop is not IPv4r2_UDP UI: %s\n", "UI sign" );
         #endif

         bf_afree(tmp_pack1);
         bf_afree(tmp_pack);
         break;
    }
    #endif

    /* unpack UDP */
    bf_afree(tmp_pack1);
    bf_afree(tmp_pack);
```

```
    data= bf_delhead(data, hdr_len + tmp_hdr_opt_len);

    #if DVI_R > 1
    printf( "read UDP frame unpacked: removed head: %u\n",
            hdr_len + tmp_hdr_opt_len );
    ip_nat_print_hex(ptr2acc_data(data), IP_MIN_HDR_SIZE, 16);
    #endif

    break;
    #endif
    /*for(;is_IPv4r2_UDP;)*/
    }

    /**/
    data_len = bf_bufsize(data);
    #if DVI_R > 1
    printf( "read incomimg bs: %u\n", data_len );
    #endif

    /* check IP hdr exist */
    #if 1
    if( data_len < IP_MIN_HDR_SIZE ){
         #if DVI_R > 1
         printf("read: no ip header\n");
         #endif

         bf_afree(data);
         return 0;
    }
```

```
#endif

/* split old ip pack to be ready to remove old ip header options */
#if 1
tmp_pack= bf_cut(data, 0, IP_MIN_HDR_SIZE);
/*
ensure we have continuous memory access to ip header
and empty tmp_pack->acc_next
*/
tmp_pack= bf_packIffLess(tmp_pack, IP_MIN_HDR_SIZE);
prev_body= tmp_pack;
tmp_pack1= tmp_pack;

ip_hdr= (ip_hdr_t *)ptr2acc_data(tmp_pack);
hdr_ver= (ip_hdr->ih_vers_ihl >> 4) & IH_IHL_MASK;
hdr_len= (ip_hdr->ih_vers_ihl & IH_IHL_MASK)*4;
ip_size= ntohs(ip_hdr->ih_length);

old_ih_src = ip_hdr->ih_src;
old_ih_dst = ip_hdr->ih_dst;
is_re_chksum= 0;
#endif

/* check IP frame is correct */
#if 1
if(
     ( hdr_ver != 4 )
     ||( hdr_len > ip_size )
     ||( ip_size > data_len)
```

```
    ){
        #if DVI_R > 1
        printf("read: wrong ip frame: "
                "hdr_ver: %u, hdr_len: %u, ip_size: %u\n",
                hdr_ver, hdr_len, ip_size);
        #endif

        bf_afree(tmp_pack);
        bf_afree(data);
        return 0;
    }
    #endif

    #if DVI_R > 1
    printf("read: before NAT: src: "); writeIpAddr(ip_hdr->ih_src);
    printf(" -> dst: "); writeIpAddr(ip_hdr->ih_dst);
    printf("\n");
    #endif

    /* split old ip header options */
    #if 1
    hdr_opt= 0;
    hdr_opt_len= hdr_len - IP_MIN_HDR_SIZE;

    #if DVI_R > 1
    printf("read opt_len: %u ip_size: %u\n", hdr_opt_len, ip_size);
    #endif

    if(hdr_opt_len)
```

```
    {
        tmp_pack= bf_cut(data, IP_MIN_HDR_SIZE, hdr_opt_len);
        /*
        ensure we have continuous memory access to ip header options
        and empty tmp_pack->acc_next
        */
        tmp_pack= bf_packIffLess(tmp_pack, hdr_opt_len);
        assert(!prev_body->acc_next);
        prev_body->acc_next= tmp_pack;
        prev_body= tmp_pack;

        hdr_opt= (u8_t *)ptr2acc_data(tmp_pack);

        #if DVI_R > 1
        ip_nat_print_hex(hdr_opt, hdr_opt_len, 16);
        #endif
    }

    old_hdr_opt= hdr_opt;
    old_hdr_opt_len= hdr_opt_len;
    #endif

    /* split ip pack body */
    #if 1
    #if DVI_R > 1
    printf( "read before delhead: %u\n", bf_bufsize(data) );
    #endif
    tmp_pack= bf_delhead(data, hdr_len);
    assert(!prev_body->acc_next);
```

```
    prev_body->acc_next= tmp_pack;
    data= tmp_pack1;
    #if DVI_R > 1
    printf( "read body: %u, hdr: %u, ip_size: %u\n", bf_bufsize(tmp_pack), hdr_len,
ip_size );
    #endif
    #endif

    /**/
    is_opt_parsed= 0;
    is_src_nat_done= 0;
    is_dst_nat_done= 0;

    /* check IPv4r2 dst addr */
    for(;is_own_IPv4r2;){
    #if 1
    /*
    else assume ih_dst is already correct if requested src NAT
    ip_hdr->ih_dst == ip_nat_wan_ip;
    */

    /* check dst is own IPv4r2 of NAT */
    /* dst IPv4 */
    if( ip_hdr->ih_dst != own_IPv4r2.ipv4 )break;

    /* check dst IPv4r2 options  */
    /* parse options */
    if(!is_opt_parsed){
    #if 1
```

```
    tmp_hdr_opt_len=
         ip_nat_walk_opt(
                   tmp_hdr_opt,
                   hdr_opt,
                   hdr_opt_len,
                   &src_IPv4r2,
                   &dst_IPv4r2
                   );
    is_opt_parsed= 1;
    #endif
    }

    /* compare dst addr */
    #if 1
    is_hdr_opt_found_B45= ip_nat_cmp_B45(&dst_IPv4r2, &own_IPv4r2);
    is_hdr_opt_found_U12= ip_nat_cmp_U12(&dst_IPv4r2, &own_IPv4r2);
    is_hdr_opt_found_V6= ip_nat_cmp_V6(&dst_IPv4r2, &own_IPv4r2);

    is_hdr_opt_found=
         is_hdr_opt_found_B45
         && is_hdr_opt_found_U12
         && is_hdr_opt_found_V6;

    #if DVI_R > 1
    printf("read dst opt found: B45: %u, U12: %u, V6: %u\n",
            is_hdr_opt_found_B45, is_hdr_opt_found_U12,
            is_hdr_opt_found_V6 );
    #endif
    #if DVI_R > 2
```

```
    ip_nat_print_hex(own_IPv4r2.b.ip_opt_b45, 8, 16);
    ip_nat_print_hex(own_IPv4r2.u.ip_opt_u12, 4, 16);
    printf("flags 0x%x\n", own_IPv4r2.ip_opt_flags );

    ip_nat_print_hex(dst_IPv4r2.b.ip_opt_b45, 8, 16);
    ip_nat_print_hex(dst_IPv4r2.u.ip_opt_u12, 4, 16);
    printf("flags 0x%x, is_parsed %u\n", dst_IPv4r2.ip_opt_flags, is_opt_parsed );
    #endif
    #endif

    /**/
    if( !is_hdr_opt_found )break;

    /* do dst NAT */
    #if 1
    is_dst_nat_done= 1;
    is_re_chksum= 1;

    ip_hdr->ih_dst= (is_IPv4r2_src)?
            /* dst NAT for specified IPv4 */
                        IPv4r2_src:
            /* dst NAT for local IPv4 */
                        ip_port->ip_ipaddr;
    #if DVI_R > 1
    printf("read dst NAT loc: "); writeIpAddr(ip_hdr->ih_dst);
    printf("\n");
    #endif
    #endif
```

```
    break;
    /*for(;is_own_IPv4r2;)*/
    #endif
    }

    /* dst NAT exist but cmp failed, try IPv4r2 via IPv4 */
    if( is_own_IPv4r2 && !is_dst_nat_done ){
    #if 1
        /* check unknown IPV4r2 can not pass via the IPv4 */
        #if 1
        if(
                /* dst NAT for specified IPv4 */
            ( is_IPv4r2_src && ( ip_hdr->ih_dst == IPv4r2_src ))
                /* dst NAT for local IPv4 */
            ||( !is_IPv4r2_src && ( ip_hdr->ih_dst == ip_port->ip_ipaddr ))
        ){
            #if DVI_R > 1
            printf( "read: drop the unknown IPv4r2 can not pass via the IPv4" );
            writeIpAddr(ip_hdr->ih_dst);
            printf( "\n" );
            ip_nat_print_hex(hdr_opt, hdr_opt_len, 16);
            #endif

            bf_afree(data);
            return 0;
        }
        #endif

        /* unknown IPV4r2 passed via the IPv4 */
```

```
        #if DVI_R > 1
        printf( "read: the unknown IPv4r2 passed via the IPv4: " );
        writeIpAddr(ip_hdr->ih_dst);
        printf( "\n" );
        ip_nat_print_hex(hdr_opt, hdr_opt_len, 16);
        #endif
    #endif
    }

    /* check IPv4r2 src addr */
    for(n=0; n<ip_nat_items; ++n){
    #if 1

    /* check IPv4 src addr */
    if( ip_nat[n].dst.ipv4 != ip_hdr->ih_src )continue;

    /* check src IPv4r2 options  */
    /* parse options */
    if(!is_opt_parsed){
    #if 1
    tmp_hdr_opt_len=
        ip_nat_walk_opt(
                tmp_hdr_opt,
                hdr_opt,
                hdr_opt_len,
                &src_IPv4r2,
                &dst_IPv4r2
                );
    is_opt_parsed= 1;
```

```
    #endif
    }

    /* compare src addr */
    #if 1
    is_hdr_opt_found_B45= ip_nat_cmp_B45(&src_IPv4r2, &ip_nat[n].dst);
    is_hdr_opt_found_U12= ip_nat_cmp_U12(&src_IPv4r2, &ip_nat[n].dst);
    is_hdr_opt_found_V6= ip_nat_cmp_V6(&src_IPv4r2, &ip_nat[n].dst);

    is_hdr_opt_found=
         is_hdr_opt_found_B45
         && is_hdr_opt_found_U12
         && is_hdr_opt_found_V6;

    #if DVI_R > 1
    printf("read src opt found: n: %u, B45: %u, U12: %u, V6: %u\n",
            n, is_hdr_opt_found_B45, is_hdr_opt_found_U12,
            is_hdr_opt_found_V6 );
    #endif
    #if DVI_R > 2
    ip_nat_print_hex(ip_nat[n].dst.b.ip_opt_b45, 8, 16);
    ip_nat_print_hex(ip_nat[n].dst.u.ip_opt_u12, 4, 16);
    printf("flags 0x%x\n", ip_nat[n].dst.ip_opt_flags );

    ip_nat_print_hex(src_IPv4r2.b.ip_opt_b45, 8, 16);
    ip_nat_print_hex(src_IPv4r2.u.ip_opt_u12, 4, 16);
    printf("flags 0x%x, is_parsed %u\n", src_IPv4r2.ip_opt_flags, is_opt_parsed );
    #endif
    #endif
```

```
    /**/
    if( !is_hdr_opt_found )continue;

    /*do src NAT*/
    #if 1
    is_src_nat_done= 1;
    is_re_chksum= 1;

    ip_hdr->ih_src = ip_nat[n].loc;

    #if DVI_R > 1
    printf("read src NAT loc: "); writeIpAddr(ip_hdr->ih_src);
    printf("\n");
    #endif
    #endif

    break;
    /*for n*/
    #endif
    }

    /* src+dst IPv4 recount UDP/TCP chksum */
    if(is_re_chksum){
    #if 1
    /* TCP/UDP chksum */
    switch(ip_hdr->ih_proto){
    #if 1
         /* TCP chksum */
```

```
        #if 1
        case 0x06:
        /* ensure we have continuous memory access to TCP header */
        prev_body->acc_next = bf_packIffLess(prev_body->acc_next, TCP_MIN_HDR_SIZE);

        tcp_hdr = (tcp_hdr_t*)ptr2acc_data(prev_body->acc_next);

        memset( &ps_hdr_tcp, 0, sizeof(ps_hdr_tcp) );
        ps_hdr_tcp.not_old_ih_src= ~old_ih_src;
        ps_hdr_tcp.not_old_ih_dst= ~old_ih_dst;
        ps_hdr_tcp.new_ih_src= ip_hdr->ih_src;
        ps_hdr_tcp.new_ih_dst= ip_hdr->ih_dst;

        tcp_hdr->th_chksum = ~oneC_sum(~(tcp_hdr->th_chksum), &ps_hdr_tcp,
sizeof(ps_hdr_tcp));
        break;
        #endif

        /* UDP chksum */
        #if 1
        case 0x11:
        /* ensure we have continuous memory access to UDP header */
        prev_body->acc_next= bf_packIffLess(prev_body->acc_next, UDP_HDR_SIZE);

        udp_hdr = (udp_hdr_t*)ptr2acc_data(prev_body->acc_next);

        #if 1
        udp_hdr->uh_chksum = 0;
```

```
        #else
        if(udp_hdr->uh_chksum){

        memset( &ps_hdr_udp, 0, sizeof(ps_hdr_udp) );
        ps_hdr_udp.not_old_ih_src= ~old_ih_src;
        ps_hdr_udp.not_old_ih_dst= ~old_ih_dst;
        ps_hdr_udp.new_ih_src= ip_hdr->ih_src;
        ps_hdr_udp.new_ih_dst= ip_hdr->ih_dst;

        udp_hdr->uh_chksum = ~oneC_sum(~(udp_hdr->uh_chksum), &ps_hdr_udp,
sizeof(ps_hdr_udp));
        }
        #endif
        break;
        #endif

    #endif
    /*switch(ip_hdr->ih_proto)*/
    }

    /*if(is_re_chksum)*/
    #endif
    }

    /* make new IPv4 options with IPv4r2 options removed */
    for(;
        is_opt_parsed
        && old_hdr_opt_len
        && ( old_hdr_opt_len != tmp_hdr_opt_len )
```

```
        ;
    ){
    #if 1
    restored_hdr_opt_len= 0;

    /*check restore src opt */
    if(!is_src_nat_done){
    #if 1
        ip_nat_buf_opt(
            restored_hdr_opt,
            &restored_hdr_opt_len,
            &src_IPv4r2
            );
    #endif
    }

    /*check restore dst opt */
    if(!is_dst_nat_done){
    #if 1
        ip_nat_buf_opt(
            restored_hdr_opt,
            &restored_hdr_opt_len,
            &dst_IPv4r2
            );
    #endif
    }

    /*check align opt in tmp */
    hdr_opt_tail = (restored_hdr_opt_len + tmp_hdr_opt_len) % 4;
```

```
    if( hdr_opt_tail ){
    #if 1
         /* align tail */
         align_opt = 0x00;
         memset(
              tmp_hdr_opt + tmp_hdr_opt_len,
              align_opt,
              4-hdr_opt_tail
              );
         tmp_hdr_opt_len += 4-hdr_opt_tail;
    #endif
    }

    /* equal size options */
    /*
    can not do the skip by orig equ translated size
    options was reparsed and realigned
    if(
         old_hdr_opt_len
         == ( restored_hdr_opt_len + tmp_hdr_opt_len )
    )break;
    */

    /* create new IP options chunk */
    hdr_opt_len= restored_hdr_opt_len + tmp_hdr_opt_len;
    #if 1
    if(!hdr_opt_len){
    /* free memory for no more options */
         assert(data->acc_next==prev_body);
```

```
        data->acc_next = prev_body->acc_next;

        tmp_pack1= prev_body;
        prev_body= data;

    }else{
    /* create new memory */
        tmp_pack= bf_memreq(hdr_opt_len);
        tmp_pack= bf_packIffLess(tmp_pack, hdr_opt_len);

        assert(data->acc_next==prev_body);
        data->acc_next = tmp_pack;
        assert(!tmp_pack->acc_next);
        tmp_pack->acc_next = prev_body->acc_next;

        tmp_pack1= prev_body;
        prev_body= tmp_pack;

        hdr_opt= (u8_t *)ptr2acc_data(tmp_pack);
        if(restored_hdr_opt_len)
            memcpy(
                hdr_opt,
                restored_hdr_opt,
                restored_hdr_opt_len
                );

        if(tmp_hdr_opt_len)
            memcpy(
                hdr_opt + restored_hdr_opt_len,
```

```
                tmp_hdr_opt,
                tmp_hdr_opt_len
                );
    }

    tmp_pack1->acc_next= 0;
    bf_afree(tmp_pack1);
    #endif

    /* correct IPv4 header */
    if( hdr_opt_len != old_hdr_opt_len ){
    #if 1
         /* new IPv4 header sizes */
         hdr_len+= hdr_opt_len-old_hdr_opt_len;
         ip_size+= hdr_opt_len-old_hdr_opt_len;

         /* do correct IPv4 header */
         ip_hdr->ih_vers_ihl&= ~IH_IHL_MASK;
         ip_hdr->ih_vers_ihl|= (hdr_len/4) & IH_IHL_MASK;
         ip_hdr->ih_length= htons(ip_size);
    #endif
    }

    break;
    /*for*/
    #endif
    }

    /* IPv4 chksum is mandatory for read */
```

```
    /* join IPv4 header buffers to simplify ip_hdr_chksum */
    #if 1
    tmp_pack= prev_body->acc_next;
    prev_body->acc_next= 0;
    #if DVI_R > 1
    printf( "read hdr bs: %u, hdr_len: %u\n", bf_bufsize(data), hdr_len );
    #endif
    assert( bf_bufsize(data) >= hdr_len );

    data = bf_pack(data);
    assert(!data->acc_next);
    data->acc_next= tmp_pack;
    #if DVI_R > 1
    printf( "read pack bs: %u, ip_size: %u\n", bf_bufsize(data), ip_size );
    #endif
    assert( bf_bufsize(data) >= ip_size );
    #endif

    /* IPv4 chksum */
    ip_hdr= (ip_hdr_t *)ptr2acc_data(data);
    ip_hdr_chksum(ip_hdr, hdr_len);

    #if DVI_R > 1
    printf("read: NAT passed: src: "); writeIpAddr(ip_hdr->ih_src);
    printf(" -> dst: "); writeIpAddr(ip_hdr->ih_dst);
    printf("\n");
    #endif

    /**/
```

```
    return data;
}
#endif

/**/
```

Все статические конфигурации вынесены в файл ip_nat_cfg.c

```
/*
ip_nat_cfg.c
*/

/* ************************************************************ */
/* NAT working mode static parameters adjustment */

/* . */
/* set static IPv4r2 NAT relations here */
static init_ip_nat_t init_ip_nat[]={
    /*
    IPv4r2 if must be the same as IPv4
    then must be placed first in the list
    oroute IPv4r2->IPv4 can be used
    to post any IPv4r2 into IPv4 dst
    */
    {"169.254.0.111",{"192.168.101.111","0/0/1","2",""}},
    {"169.254.0.121",{"192.168.101.121","0/0/2","3",""}},
    {"169.254.0.131",{"192.168.101.131","0/0/3","4",""}},
    /* examples */
    {"169.254.0.1",{"192.168.101.1","","",""}},
    {"169.254.0.11",{"192.168.101.11","","",""}},
```

```
    /*
    can not NAT to own IPv4 (own_IPv4 == 192.168.101.11),
    can map it there only (map 192.168.101.11 to 169.254.0.11)
    {"169.254.0.11",{"192.168.101.11","","",""}},
    */
    {"169.254.0.12",{"192.168.101.12","","",""}},
    {"169.254.0.13",{"192.168.101.13","","",""}}
    /*
    can not NAT to own IPv4 (own_IPv4 == 192.168.101.13),
    can map it there only (map 192.168.101.13 to 169.254.0.13)
    {"169.254.0.13",{"192.168.101.13","","",""}}
    */
};

/* . */
/* map NAT local own IPv4 into local address space of IPv4r2 NAT table */
static char is_map_own_IPv4= 1;
    /*
    do map own IPv4 specified
    just to map own IPv4 into IPv4r2 addr local area
    without IPv4r2 translation
    to work with all hosts in the same mapped local network
    by implementation
        this is the same switch src/dst addr as for 127.0.0.1
        if dst == mapped_own_IPv4
    */

/* IPv4r2 NAT table local address space is within ip_nat[].loc network */
static const char *init_mapped_own_IPv4=
```

```
    /* example */
    "169.254.0.11";

/* . */
/* NAT own IPv4 */
static char is_own_IPv4= 1;
    /*
    NAT own IPv4 (local IPv4) specified
    to simplify relations between the NAT and local networks
    we assume there is single local IPv4 only for NAT translation
    otherwise any local IPv4 will be used to do NAT
    */

static const char *init_own_IPv4=
    /* example */
    "192.168.101.11";

/* ********************************************************** */
/* . */
/* has own IPv4r2 */
static char is_own_IPv4r2= 0;
    /*
    own IPv4r2 specified
    do substitute local IPv4 to IPv4r2 specified
    src on send: to send local IPv4 as IPv4r2 addr
    dst on receive: to translate the IPv4r2 addr to local IPv4
    (NAT works with opposite addr: dst on send and src on receive)
    otherwise local IPv4 will be unchanged
    and asimmetric IPv4r2 access 'll be in use
```

```
    Note: only local IPv4 will be translated
        (ip_port->ip_ipaddr == ip_hdr->ih_src)
        otherwise we can't restore IPv4r2 to IPv4
    */

static init_ip_nat_dst_t init_own_IPv4r2=
    /* example */
    {"192.168.101.11","0/0/1","2",""};

/* . */
/* has IPv4r2 src */
static char is_IPv4r2_src= 0;
    /*
    IPv4 for IPv4r2 specified
    do substitute IPv4r2_src to IPv4r2 specified
    it is usefull for minix working as LAN bridge
    Note: only IPv4r2_src will be translated to IPv4r2
        (IPv4r2_src == ip_hdr->ih_src)
        otherwise we can't restore IPv4r2 to IPv4
    */

static const char *init_IPv4r2_src=
    /* example */
    "192.168.101.11";

/* ********************************************************** */
/* . */
/* force IPv4r2.x index format */
static char is_IPv4r2_gate_cmd= 1;
```

```
    /*
    IPv4r2.x index format specified (disable default IPv4r2 gate rules)
    set to 1 is req to IPv4r2.0 to disable IPv4r2 gate index usage
    */

/* IPv4r2 gate command 4 low bits */
static const u8_t init_IPv4r2_gate_cmd=
    /*
    set to 0 is req to IPv4r2.0 to disable IPv4r2 gate index usage
    */
    0;

/* ********************************************************** */
/* . */
/* has IPv4r2 UDP incapsulation (UI) */
static char is_IPv4r2_UDP= 0;
    /*
    UDP incapsulation for IPv4r2 frame specified
    all IPv4r2 will be packed into IPv4 UDP frame
    and all incoming UDP frames will be checked for IPv4r2 to unpack
    */

/* IPv4r2 UI magic signature */
static const u8_t IPv4r2_UDP_sign[4]=
    /* by standard */
    { 0x55, 0xaa, 0x42, 0x00 };

/* IPv4r2 UI service UDP port */
static const u16_t init_IPv4r2_UDP_port=
```

```
    /* example */
    4;

/* has IPv4r2 network entry point */
static char is_IPv4r2_UDP_dst= 0;
    /*
    IPv4r2 network IPv4 entry point specified
    send all UDP incapsulated IPv4r2 frames into the dst
    there is path to IPv4r2 network and all UDP incapsulated IPv4r2 frames
    must be sent to the entry point
    otherwise UDP incapsulated IPv4r2 frames
    will be sent to IPv4 root of IPv4r2 dst
    */

static const char *init_IPv4r2_UDP_dst=
    /* example */
    "192.88.99.1";

/* track IPv4r2 network broken path */
static char is_IPv4r2_UDP_own_src= 1;
    /*
    if set IPv4r2 UI frames will be sent from name of src of the host
    which can not deliver pure IPv4r2 frame (src == ip_port->ip_ipaddr)
    otherwise IPv4r2 origin src will be used as UI src
    in the case UI replies will be posted back to origin as UI
    */

/* allow IPv4r2 uni server addr */
static char is_IPv4r2_UDP_uni_dst= 0;
```

```
    /*
    if set IPv4r2_UDP read will accept
    dst == IPv4r2_UDP_uni_dst to do UDP->IPv4r2 decapsulation
    otherwise only own IPv4 (dst == ip_port->ip_ipaddr)
    will be used on read
    */

/* unified IPv4 addr for servers between IPv4 and IPv4r2 */
static const char *init_IPv4r2_UDP_uni_dst=
    /* example */
    "192.88.99.1";

/* unpack all UI incoming frames */
static char is_IPv4r2_UDP_read_all= 1;
    /*
    if set all incoming UDP farems with UI signature will be unpacked
    it is required for end-host if set is_own_IPv4r2
    because IPv4 part of own_IPv4r2 can be no the same as ip_port->ip_ipaddr
    the propetry is controlled separatedly to unlink with "own_IPv4r2" subsystem
    otherwise only allowed dst will be unpacked
    */

/* ********************************************************** */
/* . */
/* NAT table size */
#define IP_NAT_ITEMS 64
    /*
    while addr comparing is implemented as slow and not optimal to search
    that does not allow to use very many NAT items (about 64 max)
```

```
    because of performance penalty if the array is larger than the max
    */

/**/
```

И есть файл заголовка ip_nat.h

```
#ifndef IP_NAT_H
#define IP_NAT_H

/**/
#define IP_OPT_R2_1         0x88
#define IP_OPT_R2_2         0x8A
#define IP_OPT_R2_3         0x8B

#define IP_R2_MTU           1152
#define IP_R2_MTU_CTRL      128
#define IP_R2_MTU_DATA      1024

/**/
typedef struct ip_nat_dst {
    ipaddr_t ipv4;
    u32_t         ip_opt_flags;
    /* base or generic address space */
    /* store option first 2 bytes to work with memcpy */
    union{
        u8_t      ip_opt_b45[8];
    }b;
    /* user addr space */
    union{
```

```
        u8_t      ip_opt_u12[4];
    }u;
    /* IPv6 addr space */
    union{
        u8_t      ip_opt_v6[20];
    }v;
}ip_nat_dst_t;

typedef struct ip_nat {
    ipaddr_t      loc;
    ip_nat_dst_t  dst;
}ip_nat_t;

/* ip_opt_flags bits*/
#define IP_NAT_R2_EN_B45    (0x000001U)
#define IP_NAT_R2_EN_U12    (0x000100U)
#define IP_NAT_R2_EN_V6     (0x010000U)

/**/
typedef struct init_ip_nat_dst {
    const char         *ipv4;
    const char         *ip_opt_b45;
    const char         *ip_opt_u12;
    const char         *ip_opt_v6;
}init_ip_nat_dst_t;

typedef struct init_ip_nat {
    const char             *loc;
    init_ip_nat_dst_t  dst;
```

```
}init_ip_nat_t;

/**/
typedef struct ip_nat_ps_hdr_tcp{
     ipaddr_t not_old_ih_src;
     ipaddr_t not_old_ih_dst;
     ipaddr_t new_ih_src;
     ipaddr_t new_ih_dst;
}ip_nat_ps_hdr_tcp_t;

typedef struct ip_nat_ps_hdr_udp{
     ipaddr_t not_old_ih_src;
     ipaddr_t not_old_ih_dst;
     ipaddr_t new_ih_src;
     ipaddr_t new_ih_dst;
}ip_nat_ps_hdr_udp_t;

#endif
```

.

В файлы ip_write.c, read.c, ip.c встроены вызовы функций описанных в ip_nat.c.

-) Файл src/servers/inet/generic/ip_write.c

При написании NAT транслятора, для того чтобы в файлах /usr/etc/rc на разных машинах можно было бы делать так:

```
/usr/etc/rc
ifconfig -I /dev/ip0 -h 192.168.101.11 -n 255.255.255.0 -m 1152
add_route -o -g 192.168.101.13 -d 192.168.101.131 -n 255.255.255.255 -m 1 -I /dev/ip0

/usr/etc/rc
ifconfig -I /dev/ip0 -h 192.168.101.13 -n 255.255.255.0 -m 1152
add_route -o -g 192.168.101.11 -d 192.168.101.111 -n 255.255.255.255 -m 1 -I /dev/ip0
```

пришлось вносить множественные исправления в файл ip_write.c и небольшие изменения в файл ipr.c .

Эти исправления в части маршрутизации вызваны тем, что minix предполагает что все компьютеры локальной сети находятся до маршрутизатора в одном сегменте ethernet, т.е. к ним нельзя обратиться через какой-либо локальный компьютер, который играет в локальной сети роль шлюза. Такие ограничения понятны только для сети 169.254.0.0/16, поэтому пришлось установить приоритет для явно задаваемой в статической таблице oroute маршрутизации, поставив проверку заданного маршрута впереди маршрута по умолчанию, так что если такие ограничения все же нужны, то не нужно задавать неправильные для таких ограничений правила в таблице маршрутов.

Теперь порядок обработки маршрутов при записи IP пакетов такой:

- маршруты к самому себе;
- маршруты заданные вручную;
- маршруты по умолчанию;
- шлюз по умолчанию.

Мне не очень хотелось вносить исправления в маршрутизацию minix, но без этого выполнять IPv4 маршрутизацию в локальной сети IPv4 было бы крайне трудно, в minix это просто не было сделано и к IPv4r2 это не относится.

```
/*
ip_write.c

Copyright 1995 Philip Homburg
*/

#include "inet.h"
#include "buf.h"
#include "event.h"
#include "type.h"

#include "arp.h"
#include "assert.h"
```

```
#include "clock.h"
#include "eth.h"
#include "icmp_lib.h"
#include "io.h"
#include "ip.h"
#include "ip_int.h"
#include "ipr.h"

/*!my ip_nat.c */
EXTERN acc_t *ip_nat_write ARGS((ip_port_t *ip_port, acc_t *data));

THIS_FILE

/* debug verbouse information */
#define DVI_W 1

/**/
FORWARD void error_reply ARGS(( ip_fd_t *fd, int error ));

PUBLIC int ip_write (fd, count)
int fd;
size_t count;
{
    ip_fd_t *ip_fd;
    acc_t *pack;
    int r;

    ip_fd= &ip_fd_table[fd];
    if (count > IP_MAX_PACKSIZE)
```

```
    {
        error_reply (ip_fd, EPACKSIZE);
        return NW_OK;
    }
    pack= (*ip_fd->if_get_userdata)(ip_fd->if_srfd, (size_t)0,
        count, FALSE);
    if (!pack)
        return NW_OK;
    r= ip_send(fd, pack, count);
    assert(r != NW_WOULDBLOCK);

    if (r == NW_OK)
        error_reply (ip_fd, count);
    else
        error_reply (ip_fd, r);
    return NW_OK;
}

PUBLIC int ip_send(fd, data, data_len)
int fd;
acc_t *data;
size_t data_len;
{
    ip_port_t *ip_port;
    ip_fd_t *ip_fd;
    ip_hdr_t *ip_hdr, *tmp_hdr;
    ipaddr_t dstaddr, nexthop, hostrep_dst, my_ipaddr, netmask;
    u8_t *addrInBytes;
    acc_t *tmp_pack, *tmp_pack1;
```

```
    int hdr_len, hdr_opt_len, r, rt;
    int type, ttl;
    size_t req_mtu;
    ev_arg_t arg;

    char is_def_gw;

    ip_fd= &ip_fd_table[fd];
    ip_port= ip_fd->if_port;

    if (!(ip_fd->if_flags & IFF_OPTSET))
    {
         bf_afree(data);
         return EBADMODE;
    }

    if (!(ip_fd->if_port->ip_flags & IPF_IPADDRSET))
    {
         /* Interface is down. What kind of error do we want? For
          * the moment, we return OK.
          */
         bf_afree(data);
         return NW_OK;
    }

    data_len= bf_bufsize(data);

    if (ip_fd->if_ipopt.nwio_flags & NWIO_RWDATONLY)
    {
```

```
        tmp_pack= bf_memreq(IP_MIN_HDR_SIZE);
        tmp_pack->acc_next= data;
        data= tmp_pack;
        data_len += IP_MIN_HDR_SIZE;
    }

    if (data_len<IP_MIN_HDR_SIZE)
    {
        bf_afree(data);
        return EPACKSIZE;
    }

    /* make sure continuous memory access to header */
    data= bf_packIffLess(data, IP_MIN_HDR_SIZE);
    ip_hdr= (ip_hdr_t *)ptr2acc_data(data);

    /* avoid shared access to header */
    if (data->acc_linkC != 1 || data->acc_buffer->buf_linkC != 1)
    {
        tmp_pack= bf_memreq(IP_MIN_HDR_SIZE);
        tmp_hdr= (ip_hdr_t *)ptr2acc_data(tmp_pack);
        *tmp_hdr= *ip_hdr;
        tmp_pack->acc_next= bf_cut(data, IP_MIN_HDR_SIZE,
            data_len-IP_MIN_HDR_SIZE);
        bf_afree(data);
        ip_hdr= tmp_hdr;
        data= tmp_pack;
        assert (data->acc_length >= IP_MIN_HDR_SIZE);
    }
```

```
if (ip_fd->if_ipopt.nwio_flags & NWIO_HDR_O_SPEC)
{
     hdr_opt_len= ip_fd->if_ipopt.nwio_hdropt.iho_opt_siz;
     if (hdr_opt_len)
     {
          tmp_pack= bf_cut(data, 0, IP_MIN_HDR_SIZE);
          tmp_pack1= bf_cut (data, IP_MIN_HDR_SIZE,
               data_len-IP_MIN_HDR_SIZE);
          bf_afree(data);
          /*!my*/
          data= bf_packIffLess(tmp_pack, IP_MIN_HDR_SIZE);
          tmp_pack= bf_memreq (hdr_opt_len);
          tmp_pack= bf_packIffLess(tmp_pack, hdr_opt_len);
          memcpy (ptr2acc_data(tmp_pack), ip_fd->if_ipopt.
               nwio_hdropt.iho_data, hdr_opt_len);

          data->acc_next= tmp_pack;
          tmp_pack->acc_next= tmp_pack1;
          hdr_len= IP_MIN_HDR_SIZE+hdr_opt_len;
     }
     else
          hdr_len= IP_MIN_HDR_SIZE;

     /*!my*/
     data= bf_packIffLess(data, hdr_len);
     ip_hdr= (ip_hdr_t *)ptr2acc_data(data);

     /**/
```

```
        ip_hdr->ih_vers_ihl= hdr_len/4;
        ip_hdr->ih_tos= ip_fd->if_ipopt.nwio_tos;
        ip_hdr->ih_flags_fragoff= 0;
        if (ip_fd->if_ipopt.nwio_df)
            ip_hdr->ih_flags_fragoff |= HTONS(IH_DONT_FRAG);
        ip_hdr->ih_ttl= ip_fd->if_ipopt.nwio_ttl;
        ttl= ORTD_UNREACHABLE+1;        /* Don't check TTL */
    }
    else
    {
        hdr_len= (ip_hdr->ih_vers_ihl & IH_IHL_MASK)*4;
        r= NW_OK;
        if (hdr_len<IP_MIN_HDR_SIZE)
            r= EINVAL;
        else if (hdr_len>data_len)
            r= EPACKSIZE;
        else if (!ip_hdr->ih_ttl)
            r= EINVAL;

        if (r != NW_OK)
        {
            bf_afree(data);
            return r;
        }

        data= bf_packIffLess(data, hdr_len);
        ip_hdr= (ip_hdr_t *)ptr2acc_data(data);
        if (hdr_len != IP_MIN_HDR_SIZE)
        {
```

```
            r= ip_chk_hdropt((u8_t *)(ptr2acc_data(data) +
                 IP_MIN_HDR_SIZE),
                 hdr_len-IP_MIN_HDR_SIZE);
            if (r != NW_OK)
            {
                 bf_afree(data);
                 return r;
            }
        }
        ttl= ip_hdr->ih_ttl;
    }

    /*!my*/
    if (ip_fd->if_ipopt.nwio_flags & NWIO_HDRANY){}
    else{
    ip_hdr->ih_vers_ihl=
         (ip_hdr->ih_vers_ihl & IH_IHL_MASK)
         | (IP_VERSION << 4);
    ip_hdr->ih_length= htons(data_len);
    if (ip_fd->if_ipopt.nwio_flags & NWIO_PROTOSPEC)
         ip_hdr->ih_proto= ip_fd->if_ipopt.nwio_proto;
    if (ip_fd->if_ipopt.nwio_flags & NWIO_REMSPEC)
         ip_hdr->ih_dst= ip_fd->if_ipopt.nwio_rem;
    if (! (ip_fd->if_ipopt.nwio_flags & NWIO_SRCANY) )
         ip_hdr->ih_src= ip_fd->if_port->ip_ipaddr;
    if (! (ip_fd->if_ipopt.nwio_flags & NWIO_IDANY) ){
         ip_hdr->ih_flags_fragoff &=     ~HTONS(
              IH_FRAGOFF_MASK
              | IH_FLAGS_UNUSED
```

```
            | IH_MORE_FRAGS
            );
        ip_hdr->ih_id= htons(ip_port->ip_frame_id++);
        }
    /*!NWIO_HDRANY*/
    }

    /*!my NAT*/
    #if DVI_W > 1
    printf("write: before NAT: dst: "); writeIpAddr(ip_hdr->ih_dst);
    printf(" <- src: "); writeIpAddr(ip_hdr->ih_src);
    printf("\n");
    #endif

    data = ip_nat_write(ip_port, data);
    if(!data){ return EPACKSIZE; }

    ip_hdr= (ip_hdr_t *)ptr2acc_data(data);
    #if DVI_W > 1
    printf("write: NAT passed: dst: "); writeIpAddr(ip_hdr->ih_dst);
    printf(" <- src: "); writeIpAddr(ip_hdr->ih_src);
    printf("\n");
    #endif

    hdr_len= (ip_hdr->ih_vers_ihl & IH_IHL_MASK)*4;
    ip_hdr_chksum(ip_hdr, hdr_len);

    /* ********** */
    /* !my oroute */
```

```
    netmask= ip_port->ip_subnetmask;
    my_ipaddr= ip_port->ip_ipaddr;

    dstaddr= ip_hdr->ih_dst;
    hostrep_dst= ntohl(dstaddr);

    /* chk mtu */
    if (ip_hdr->ih_flags_fragoff & HTONS(IH_DONT_FRAG))
    {
         req_mtu= bf_bufsize(data);
         if (req_mtu > ip_port->ip_mtu)
         {
              DBLOCK(1, printf(
              "packet is larger than link MTU and DF is set\n"));
              bf_afree(data);
              return EPACKSIZE;
         }
    }
    else
         req_mtu= 0;

    /* local loopback */
    addrInBytes= (u8_t *)&dstaddr;
    if ((addrInBytes[0] & 0xff) == 0x7f)
    {
         assert (data->acc_linkC == 1);
         dstaddr= ip_hdr->ih_dst;    /* swap src and dst
                                * addresses */
         ip_hdr->ih_dst= ip_hdr->ih_src;
```

```
        ip_hdr->ih_src= dstaddr;
        data->acc_ext_link= NULL;
        if (ip_port->ip_loopb_head == NULL)
        {
             ip_port->ip_loopb_head= data;
             arg.ev_ptr= ip_port;
             ev_enqueue(&ip_port->ip_loopb_event,
                  ip_process_loopb, arg);
        }
        else
             ip_port->ip_loopb_tail->acc_ext_link= data;
        ip_port->ip_loopb_tail= data;

        return NW_OK;
    }

    if (dstaddr == my_ipaddr)
    {
        assert (data->acc_linkC == 1);

        data->acc_ext_link= NULL;
        if (ip_port->ip_loopb_head == NULL)
        {
             ip_port->ip_loopb_head= data;
             arg.ev_ptr= ip_port;
             ev_enqueue(&ip_port->ip_loopb_event,
                  ip_process_loopb, arg);
        }
        else
```

```
            ip_port->ip_loopb_tail->acc_ext_link= data;
        ip_port->ip_loopb_tail= data;

        return NW_OK;
    }

    /* oroute static table */
    rt = oroute_frag (ip_port - ip_port_table, dstaddr, ttl, req_mtu,
        &nexthop, &is_def_gw );

    #if 1
        printf("dstaddr: "); writeIpAddr(dstaddr);
        printf(" nexthop: "); writeIpAddr(nexthop);
        printf(" =%s\n", ((is_def_gw)?"*def_gw":"expl_gw") );
        printf("(my_ip= ");writeIpAddr(my_ipaddr);
        printf(", ip_port= "); writeIpAddr(ip_port->ip_ipaddr);
        printf(", got rt=0x%x /0 is OK/)\n", rt);
    #endif

    /* found explicit oroute */
    if( (rt == NW_OK) && !is_def_gw )
    {
        /* local loopback */
        if (nexthop == ip_port->ip_ipaddr)
        {
            data->acc_ext_link= NULL;
            if (ip_port->ip_loopb_head == NULL)
            {
                ip_port->ip_loopb_head= data;
```

```
                arg.ev_ptr= ip_port;
                ev_enqueue(&ip_port->ip_loopb_event,
                    ip_process_loopb, arg);
            }
            else
                ip_port->ip_loopb_tail->acc_ext_link= data;
            ip_port->ip_loopb_tail= data;
        }
        /* explicit oroute */
        else
        {
            rt= (*ip_port->ip_dev_send)(ip_port,
                nexthop, data, IP_LT_NORMAL);
        }

        return rt;
    }

    /* default oroute */
    r= 0;
    if (hostrep_dst == (ipaddr_t)-1)
        ;   /* OK, local broadcast */
    else if ((hostrep_dst & 0xe0000000l) == 0xe0000000l)
        ;   /* OK, Multicast */
    else if ((hostrep_dst & 0xf0000000l) == 0xf0000000l)
        r= EBADDEST;  /* Bad class */
    else if ((dstaddr ^ my_ipaddr) & netmask)
        ;   /* OK, remote destination (not the local IP network) */
    else if (
```

```
            !(dstaddr & ~netmask)  /* Zero host part */
            && (ip_port->ip_flags & IPF_SUBNET_BCAST)
    ){
        r= EBADDEST;  /* Zero host part are not enabled as host */
    }

    if (r<0)
    {
        DIFBLOCK(1, r == EBADDEST,
            printf("bad destination: ");
            writeIpAddr(ip_hdr->ih_dst);
            printf("\n"));
        bf_afree(data);
        return r;
    }

    /* BCAST, MCAST */
    if ((dstaddr & HTONL(0xe0000000)) == HTONL(0xe0000000))
    {
        if (dstaddr == (ipaddr_t)-1)
        {
            r= (*ip_port->ip_dev_send)(ip_port, dstaddr, data,
                IP_LT_BROADCAST);
            return r;
        }
        if (ip_nettype(dstaddr) == IPNT_CLASS_D)
        {
            /* Multicast, what about multicast routing? */
            r= (*ip_port->ip_dev_send)(ip_port, dstaddr, data,
```

```
                IP_LT_MULTICAST);
            return r;
        }
    }

    /* local network */
    if (((dstaddr ^ my_ipaddr) & netmask) == 0)
    {
        type= ((dstaddr == (my_ipaddr | ~netmask) &&
            (ip_port->ip_flags & IPF_SUBNET_BCAST)) ?
            IP_LT_BROADCAST : IP_LT_NORMAL);

        r= (*ip_port->ip_dev_send)(ip_port, dstaddr, data, type);

        /*!my*/
        #if 0
        printf(
        "dstaddr %x:\n(my= %x; ip_port= %x; r= %x)\n",
        dstaddr, my_ipaddr, ip_port->ip_ipaddr, r );
        #endif
        return r;
    }

    /* there is default gateway */
    if( rt == NW_OK )
    {
        rt= (*ip_port->ip_dev_send)(ip_port,
                nexthop, data, IP_LT_NORMAL);
        return rt;
```

```
    }

    /* no oroute */
    DBLOCK(0x10, printf("got error %d\n", rt));
    bf_afree(data);
    return rt;
}

PUBLIC void ip_hdr_chksum(ip_hdr, ip_hdr_len)
ip_hdr_t *ip_hdr;
int ip_hdr_len;
{
    ip_hdr->ih_hdr_chk= 0;
    ip_hdr->ih_hdr_chk= ~oneC_sum (0, (u16_t *)ip_hdr, ip_hdr_len);
}

PUBLIC acc_t *ip_split_pack (ip_port, ref_last, mtu)
ip_port_t *ip_port;
acc_t **ref_last;
int mtu;
{
    int pack_siz;
    ip_hdr_t *first_hdr, *second_hdr;
    int first_hdr_len, second_hdr_len;
    int first_data_len, second_data_len;
    int data_len, max_data_len, nfrags, new_first_data_len;
    int first_opt_size, second_opt_size;
    acc_t *first_pack, *second_pack, *tmp_pack;
    u8_t *first_optptr, *second_optptr;
```

```
    int i, optlen;

    first_pack= *ref_last;
    *ref_last= 0;
    second_pack= 0;

    first_pack= bf_align(first_pack, IP_MIN_HDR_SIZE, 4);
    first_pack= bf_packIffLess(first_pack, IP_MIN_HDR_SIZE);
    assert (first_pack->acc_length >= IP_MIN_HDR_SIZE);

    first_hdr= (ip_hdr_t *)ptr2acc_data(first_pack);
    first_hdr_len= (first_hdr->ih_vers_ihl & IH_IHL_MASK) * 4;
    if (first_hdr_len>IP_MIN_HDR_SIZE)
    {
        first_pack= bf_packIffLess(first_pack, first_hdr_len);
        first_hdr= (ip_hdr_t *)ptr2acc_data(first_pack);
    }

    pack_siz= bf_bufsize(first_pack);
    assert(pack_siz > mtu);

    assert (!(first_hdr->ih_flags_fragoff & HTONS(IH_DONT_FRAG)));

    if (first_pack->acc_linkC != 1 ||
        first_pack->acc_buffer->buf_linkC != 1)
    {
        /* Get a private copy of the IP header */
        tmp_pack= bf_memreq(first_hdr_len);
        memcpy(ptr2acc_data(tmp_pack), first_hdr, first_hdr_len);
```

```
        first_pack= bf_delhead(first_pack, first_hdr_len);
        tmp_pack->acc_next= first_pack;
        first_pack= tmp_pack; tmp_pack= NULL;
        first_hdr= (ip_hdr_t *)ptr2acc_data(first_pack);
    }

    data_len= ntohs(first_hdr->ih_length) - first_hdr_len;

    /* Try to split the packet evenly. */
    assert(mtu > first_hdr_len);
    max_data_len= mtu-first_hdr_len;
    nfrags= (data_len/max_data_len)+1;
    new_first_data_len= data_len/nfrags;
    if (new_first_data_len < 8)
    {
        /* Special case for extremely small MTUs */
        new_first_data_len= 8;
    }
    new_first_data_len &= ~7; /* data goes in 8 byte chuncks */

    assert(new_first_data_len >= 8);
    assert(new_first_data_len+first_hdr_len <= mtu);

    second_data_len= data_len-new_first_data_len;
    second_pack= bf_cut(first_pack, first_hdr_len+
        new_first_data_len, second_data_len);
    tmp_pack= first_pack;
    first_data_len= new_first_data_len;
    first_pack= bf_cut (tmp_pack, 0, first_hdr_len+first_data_len);
```

```
    bf_afree(tmp_pack);
    tmp_pack= bf_memreq(first_hdr_len);
    tmp_pack->acc_next= second_pack;
    second_pack= tmp_pack;
    second_hdr= (ip_hdr_t *)ptr2acc_data(second_pack);
    *second_hdr= *first_hdr;
    second_hdr->ih_flags_fragoff= htons(
         ntohs(first_hdr->ih_flags_fragoff)+(first_data_len/8));

    first_opt_size= first_hdr_len-IP_MIN_HDR_SIZE;
    second_opt_size= 0;
    if (first_opt_size)
    {
         first_pack= bf_packIffLess (first_pack,
              first_hdr_len);
         first_hdr= (ip_hdr_t *)ptr2acc_data(first_pack);
         assert (first_pack->acc_length>=first_hdr_len);
         first_optptr= (u8_t *)ptr2acc_data(first_pack)+
              IP_MIN_HDR_SIZE;
         second_optptr= (u8_t *)ptr2acc_data(
              second_pack)+IP_MIN_HDR_SIZE;
         i= 0;
         while (i<first_opt_size)
         {
              switch (*first_optptr & IP_OPT_NUMBER)
              {
              case 0:
              case 1:
                   optlen= 1;
```

```
                    break;
               default:
                    optlen= first_optptr[1];
                    break;
               }
               assert (i + optlen <= first_opt_size);
               i += optlen;
               if (*first_optptr & IP_OPT_COPIED)
               {
                    second_opt_size += optlen;
                    while (optlen--)
                         *second_optptr++=
                              *first_optptr++;
               }
               else
                    first_optptr += optlen;
          }
          while (second_opt_size & 3)
          {
               *second_optptr++= 0;
               second_opt_size++;
          }
     }
     second_hdr_len= IP_MIN_HDR_SIZE + second_opt_size;

     second_hdr->ih_vers_ihl= (second_hdr->ih_vers_ihl & 0xf0)
          + (second_hdr_len/4);
     second_hdr->ih_length= htons(second_data_len+
          second_hdr_len);
```

```
    second_pack->acc_length= second_hdr_len;

    assert(first_pack->acc_linkC == 1);
    assert(first_pack->acc_buffer->buf_linkC == 1);

    first_hdr->ih_flags_fragoff |= HTONS(IH_MORE_FRAGS);
    first_hdr->ih_length= htons(first_data_len+
        first_hdr_len);
    assert (!(second_hdr->ih_flags_fragoff & HTONS(IH_DONT_FRAG)));

    ip_hdr_chksum(first_hdr, first_hdr_len);
    if (second_data_len+second_hdr_len <= mtu)
    {
        /* second_pack will not be split any further, so we have to
         * calculate the header checksum.
         */
        ip_hdr_chksum(second_hdr, second_hdr_len);
    }

    *ref_last= second_pack;

    return first_pack;
}

PRIVATE void error_reply (ip_fd, error)
ip_fd_t *ip_fd;
int error;
{
    if ((*ip_fd->if_get_userdata)(ip_fd->if_srfd, (size_t)error,
```

```
            (size_t)0, FALSE))
        {
            ip_panic(( "can't error_reply" ));
        }
}

/*
 * $PchId: ip_write.c,v 1.22 2004/08/03 11:11:04 philip Exp $
 */
```

.

-) Файл src/servers/inet/generic/ip_read.c

Изменения в этот файл:

```
/*
ip_read.c

Copyright 1995 Philip Homburg
*/

#include "inet.h"
#include "buf.h"
#include "clock.h"
#include "event.h"
#include "type.h"

#include "assert.h"
#include "icmp_lib.h"
#include "io.h"
#include "ip.h"
```

```
#include "ip_int.h"
#include "ipr.h"

/*!my ip_nat.c */
EXTERN acc_t *ip_nat_read ARGS((ip_port_t *ip_port, acc_t *data));

THIS_FILE

/* debug verbouse information */
#define DVI_R 1

/*!my */
PUBLIC void ip_loopb_arrived(ip_port_t *ip_port, acc_t *pack);

FORWARD ip_ass_t *find_ass_ent ARGS(( ip_port_t *ip_port, U16_t id,
    int proto, ipaddr_t src, ipaddr_t dst ));
FORWARD acc_t *merge_frags ARGS(( acc_t *first, acc_t *second ));
FORWARD int ip_frag_chk ARGS(( acc_t *pack ));
FORWARD acc_t *reassemble ARGS(( ip_port_t *ip_port, acc_t *pack,
    ip_hdr_t *ip_hdr ));
FORWARD void route_packets ARGS(( event_t *ev, ev_arg_t ev_arg ));
FORWARD int broadcast_dst ARGS(( ip_port_t *ip_port, ipaddr_t dest ));

PUBLIC int ip_read (fd, count)
int fd;
size_t count;
{
    ip_fd_t *ip_fd;
    acc_t *pack;
```

```
ip_fd= &ip_fd_table[fd];
if (!(ip_fd->if_flags & IFF_OPTSET))
{
     return (*ip_fd->if_put_userdata)(ip_fd->if_srfd, EBADMODE,
          (acc_t *)0, FALSE);
}

ip_fd->if_rd_count= count;

ip_fd->if_flags |= IFF_READ_IP;
if (ip_fd->if_rdbuf_head)
{
     if (get_time() <= ip_fd->if_exp_time)
     {
          pack= ip_fd->if_rdbuf_head;
          ip_fd->if_rdbuf_head= pack->acc_ext_link;
          ip_packet2user (ip_fd, pack, ip_fd->if_exp_time,
               bf_bufsize(pack));
          assert(!(ip_fd->if_flags & IFF_READ_IP));
          return NW_OK;
     }
     while (ip_fd->if_rdbuf_head)
     {
          pack= ip_fd->if_rdbuf_head;
          ip_fd->if_rdbuf_head= pack->acc_ext_link;
          bf_afree(pack);
     }
}
```

```
    return NW_SUSPEND;
}

PRIVATE acc_t *reassemble (ip_port, pack, pack_hdr)
ip_port_t *ip_port;
acc_t *pack;
ip_hdr_t *pack_hdr;
{
    ip_ass_t *ass_ent;
    size_t pack_hdr_len, pack_data_len, pack_offset, tmp_offset;
    u16_t pack_flags_fragoff;
    acc_t *prev_acc, *curr_acc, *next_acc, *head_acc, *tmp_acc;
    ip_hdr_t *tmp_hdr;
    time_t first_time;

    ass_ent= find_ass_ent (ip_port, pack_hdr->ih_id,
        pack_hdr->ih_proto, pack_hdr->ih_src, pack_hdr->ih_dst);

    pack_flags_fragoff= ntohs(pack_hdr->ih_flags_fragoff);
    pack_hdr_len= (pack_hdr->ih_vers_ihl & IH_IHL_MASK) * 4;
    pack_data_len= ntohs(pack_hdr->ih_length)-pack_hdr_len;
    pack_offset= (pack_flags_fragoff & IH_FRAGOFF_MASK)*8;
    pack->acc_ext_link= NULL;

    head_acc= ass_ent->ia_frags;
    ass_ent->ia_frags= NULL;
    if (head_acc == NULL)
    {
        ass_ent->ia_frags= pack;
```

```
        return NULL;
    }

    prev_acc= NULL;
    curr_acc= NULL;
    next_acc= head_acc;

    while(next_acc)
    {
        tmp_hdr= (ip_hdr_t *)ptr2acc_data(next_acc);
        tmp_offset= (ntohs(tmp_hdr->ih_flags_fragoff) &
            IH_FRAGOFF_MASK)*8;

        if (pack_offset < tmp_offset)
            break;

        prev_acc= curr_acc;
        curr_acc= next_acc;
        next_acc= next_acc->acc_ext_link;
    }
    if (curr_acc == NULL)
    {
        assert(prev_acc == NULL);
        assert(next_acc != NULL);

        curr_acc= merge_frags(pack, next_acc);
        head_acc= curr_acc;
    }
    else
```

```
    {
         curr_acc= merge_frags(curr_acc, pack);
         if (next_acc != NULL)
              curr_acc= merge_frags(curr_acc, next_acc);
         if (prev_acc != NULL)
              prev_acc->acc_ext_link= curr_acc;
         else
              head_acc= curr_acc;
    }
    ass_ent->ia_frags= head_acc;

    pack= ass_ent->ia_frags;
    pack_hdr= (ip_hdr_t *)ptr2acc_data(pack);
    pack_flags_fragoff= ntohs(pack_hdr->ih_flags_fragoff);

    if (!(pack_flags_fragoff & (IH_FRAGOFF_MASK|IH_MORE_FRAGS)))
         /* it's now a complete packet */
    {
         first_time= ass_ent->ia_first_time;

         ass_ent->ia_frags= 0;
         ass_ent->ia_first_time= 0;

         while (pack->acc_ext_link)
         {
              tmp_acc= pack->acc_ext_link;
              pack->acc_ext_link= tmp_acc->acc_ext_link;
              bf_afree(tmp_acc);
         }
```

```
        if ((ass_ent->ia_min_ttl) * HZ + first_time <
            get_time())
        {
            if (broadcast_dst(ip_port, pack_hdr->ih_dst))
            {
                DBLOCK(1, printf(
    "ip_read'reassemble: reassembly timeout for broadcast packet\n"););
                bf_afree(pack); pack= NULL;
                return NULL;
            }
            icmp_snd_time_exceeded(ip_port->ip_port, pack,
                ICMP_FRAG_REASSEM);
        }
        else
            return pack;
    }
    return NULL;
}

PRIVATE acc_t *merge_frags (first, second)
acc_t *first, *second;
{
    ip_hdr_t *first_hdr, *second_hdr;
    size_t first_hdr_size, second_hdr_size, first_datasize, second_datasize,
        first_offset, second_offset;
    acc_t *cut_second, *tmp_acc;

    if (!second)
    {
```

```
        first->acc_ext_link= NULL;
        return first;
    }

assert (first->acc_length >= IP_MIN_HDR_SIZE);
assert (second->acc_length >= IP_MIN_HDR_SIZE);

    first_hdr= (ip_hdr_t *)ptr2acc_data(first);
    first_offset= (ntohs(first_hdr->ih_flags_fragoff) &
        IH_FRAGOFF_MASK) * 8;
    first_hdr_size= (first_hdr->ih_vers_ihl & IH_IHL_MASK) * 4;
    first_datasize= ntohs(first_hdr->ih_length) - first_hdr_size;

    second_hdr= (ip_hdr_t *)ptr2acc_data(second);
    second_offset= (ntohs(second_hdr->ih_flags_fragoff) &
        IH_FRAGOFF_MASK) * 8;
    second_hdr_size= (second_hdr->ih_vers_ihl & IH_IHL_MASK) * 4;
    second_datasize= ntohs(second_hdr->ih_length) - second_hdr_size;

    assert (first_hdr_size + first_datasize == bf_bufsize(first));
    assert (second_hdr_size + second_datasize == bf_bufsize(second));
    assert (second_offset >= first_offset);

    if (second_offset > first_offset+first_datasize)
    {
        DBLOCK(1, printf("ip fragments out of order\n"));
        first->acc_ext_link= second;
        return first;
    }
```

```
    if (second_offset + second_datasize <= first_offset +
        first_datasize)
    {
        /* May cause problems if we try to merge. */
        bf_afree(first);
        return second;
    }

    if (!(second_hdr->ih_flags_fragoff & HTONS(IH_MORE_FRAGS)))
        first_hdr->ih_flags_fragoff &= ~HTONS(IH_MORE_FRAGS);

    second_datasize= second_offset+second_datasize-(first_offset+
        first_datasize);
    cut_second= bf_cut(second, second_hdr_size + first_offset+
        first_datasize-second_offset, second_datasize);
    tmp_acc= second->acc_ext_link;
    bf_afree(second);
    second= tmp_acc;

    first_datasize += second_datasize;
    first_hdr->ih_length= htons(first_hdr_size + first_datasize);

    first= bf_append (first, cut_second);
    first->acc_ext_link= second;

assert (first_hdr_size + first_datasize == bf_bufsize(first));

    return first;
```

```
}

PRIVATE ip_ass_t *find_ass_ent (ip_port, id, proto, src, dst)
ip_port_t *ip_port;
u16_t id;
ipproto_t proto;
ipaddr_t src;
ipaddr_t dst;
{
     ip_ass_t *new_ass_ent, *tmp_ass_ent;
     int i;
     acc_t *tmp_acc, *curr_acc;

     new_ass_ent= 0;

     for (i=0, tmp_ass_ent= ip_ass_table; i<IP_ASS_NR; i++,
          tmp_ass_ent++)
     {
          if (!tmp_ass_ent->ia_frags && tmp_ass_ent->ia_first_time)
          {
               DBLOCK(1,
          printf("strange ip_ass entry (can be a race condition)\n"));
               continue;
          }

          if ((tmp_ass_ent->ia_srcaddr == src) &&
               (tmp_ass_ent->ia_dstaddr == dst) &&
               (tmp_ass_ent->ia_proto == proto) &&
               (tmp_ass_ent->ia_id == id) &&
```

```
            (tmp_ass_ent->ia_port == ip_port))
        {
            return tmp_ass_ent;
        }
        if (!new_ass_ent || tmp_ass_ent->ia_first_time <
            new_ass_ent->ia_first_time)
        {
            new_ass_ent= tmp_ass_ent;
        }
    }

    if (new_ass_ent->ia_frags)
    {
        DBLOCK(2, printf("old frags id= %u, proto= %u, src= ",
            ntohs(new_ass_ent->ia_id),
            new_ass_ent->ia_proto);
            writeIpAddr(new_ass_ent->ia_srcaddr); printf(" dst= ");
            writeIpAddr(new_ass_ent->ia_dstaddr); printf(": ");
            ip_print_frags(new_ass_ent->ia_frags); printf("\n"));
        curr_acc= new_ass_ent->ia_frags->acc_ext_link;
        while (curr_acc)
        {
            tmp_acc= curr_acc->acc_ext_link;
            bf_afree(curr_acc);
            curr_acc= tmp_acc;
        }
        curr_acc= new_ass_ent->ia_frags;
        new_ass_ent->ia_frags= 0;
        if (broadcast_dst(ip_port, new_ass_ent->ia_dstaddr))
```

```
        {
            DBLOCK(1, printf(
    "ip_read'find_ass_ent: reassembly timeout for broadcast packet\n"));
            bf_afree(curr_acc); curr_acc= NULL;
        }
        else
        {
            icmp_snd_time_exceeded(ip_port->ip_port,
                curr_acc, ICMP_FRAG_REASSEM);
        }
    }
    new_ass_ent->ia_min_ttl= IP_MAX_TTL;
    new_ass_ent->ia_port= ip_port;
    new_ass_ent->ia_first_time= get_time();
    new_ass_ent->ia_srcaddr= src;
    new_ass_ent->ia_dstaddr= dst;
    new_ass_ent->ia_proto= proto;
    new_ass_ent->ia_id= id;

    return new_ass_ent;
}

PRIVATE int ip_frag_chk(pack)
acc_t *pack;
{
    ip_hdr_t *ip_hdr;
    int hdr_len;

    if (pack->acc_length < sizeof(ip_hdr_t))
```

```
    {
        #if DVI_R > 1
        printf("frag error: non flat mem: size IP_MIN_HDR_SIZE\n" );
        #endif

        DBLOCK(1, printf("wrong length\n"));
        return FALSE;
    }

    ip_hdr= (ip_hdr_t *)ptr2acc_data(pack);

    hdr_len= (ip_hdr->ih_vers_ihl & IH_IHL_MASK) * 4;
    if (pack->acc_length < hdr_len)
    {
        #if DVI_R > 1
        printf("frag error: non flat mem: size hdr_len\n" );
        #endif

        DBLOCK(1, printf("wrong length\n"));
        return FALSE;
    }

    if (((ip_hdr->ih_vers_ihl >> 4) & IH_VERSION_MASK) !=
        IP_VERSION)
    {
        #if DVI_R > 1
        printf("frag error: wrong hdr ver 0x%x\n", ip_hdr->ih_vers_ihl );
        #endif
```

```
        DBLOCK(1, printf("wrong version (ih_vers_ihl=0x%x)\n",
             ip_hdr->ih_vers_ihl));
        return FALSE;
    }
    /*!my: was != bf_bufsize(pack) */
    if (ntohs(ip_hdr->ih_length) > bf_bufsize(pack))
    {
        #if DVI_R > 1
        printf("frag error: ip size %u > mem size %u\n",
                ip_hdr->ih_length, bf_bufsize(pack) );
        #endif

        DBLOCK(1, printf("wrong size\n"));
        return FALSE;
    }
    if ((u16_t)~oneC_sum(0, (u16_t *)ip_hdr, hdr_len))
    {
        #if DVI_R > 1
        printf("frag error: ip hdr chksum\n");
        #endif

        DBLOCK(1, printf("packet with wrong checksum (= %x)\n",
             (u16_t)~oneC_sum(0, (u16_t *)ip_hdr, hdr_len)));
        return FALSE;
    }
    if (hdr_len>IP_MIN_HDR_SIZE && ip_chk_hdropt((u8_t *)
        (ptr2acc_data(pack) + IP_MIN_HDR_SIZE),
        hdr_len-IP_MIN_HDR_SIZE))
    {
```

```
            #if DVI_R > 1
            printf("frag error: wrong ip options\n");
            #endif

            DBLOCK(1, printf("packet with wrong options\n"));
            return FALSE;
        }
        return TRUE;
}

PUBLIC void ip_packet2user (ip_fd, pack, exp_time, data_len)
ip_fd_t *ip_fd;
acc_t *pack;
time_t exp_time;
size_t data_len;
{
        acc_t *tmp_pack;
        ip_hdr_t *ip_hdr;
        int result, ip_hdr_len;
        size_t transf_size;

        assert (ip_fd->if_flags & IFF_INUSE);
        if (!(ip_fd->if_flags & IFF_READ_IP))
        {
            if (pack->acc_linkC != 1)
            {
                tmp_pack= bf_dupacc(pack);
                bf_afree(pack);
                pack= tmp_pack;
```

```
            tmp_pack= NULL;
        }
        pack->acc_ext_link= NULL;
        if (ip_fd->if_rdbuf_head == NULL)
        {
            ip_fd->if_rdbuf_head= pack;
            ip_fd->if_exp_time= exp_time;
        }
        else
            ip_fd->if_rdbuf_tail->acc_ext_link= pack;
        ip_fd->if_rdbuf_tail= pack;
        return;
    }

    assert (pack->acc_length >= IP_MIN_HDR_SIZE);
    ip_hdr= (ip_hdr_t *)ptr2acc_data(pack);

    if (ip_fd->if_ipopt.nwio_flags & NWIO_RWDATONLY)
    {
        ip_hdr_len= (ip_hdr->ih_vers_ihl & IH_IHL_MASK) * 4;

        assert (data_len > ip_hdr_len);
        data_len -= ip_hdr_len;
        pack= bf_delhead(pack, ip_hdr_len);
    }

    if (data_len > ip_fd->if_rd_count)
    {
        tmp_pack= bf_cut (pack, 0, ip_fd->if_rd_count);
```

```
            bf_afree(pack);
            pack= tmp_pack;
            transf_size= ip_fd->if_rd_count;
        }
        else
            transf_size= data_len;

        if (ip_fd->if_put_pkt)
        {
            (*ip_fd->if_put_pkt)(ip_fd->if_srfd, pack, transf_size);
            return;
        }

        result= (*ip_fd->if_put_userdata)(ip_fd->if_srfd,
            (size_t)0, pack, FALSE);
        if (result >= 0)
        {
            if (data_len > transf_size)
                result= EPACKSIZE;
            else
                result= transf_size;
        }

        ip_fd->if_flags &= ~IFF_READ_IP;
        result= (*ip_fd->if_put_userdata)(ip_fd->if_srfd, result,
                (acc_t *)0, FALSE);
        assert (result >= 0);
}
```

```
PUBLIC void ip_port_arrive (ip_port, pack, ip_hdr)
ip_port_t *ip_port;
acc_t *pack;
ip_hdr_t *ip_hdr;
{
     ip_fd_t *ip_fd, *first_fd, *share_fd;
     unsigned long ip_pack_stat;
     unsigned size;
     int i;
     int hash, proto;
     time_t exp_time;

     assert (pack->acc_linkC>0);
     assert (pack->acc_length >= IP_MIN_HDR_SIZE);

     if (ntohs(ip_hdr->ih_flags_fragoff) & (IH_FRAGOFF_MASK|IH_MORE_FRAGS))
     {
          pack= reassemble (ip_port, pack, ip_hdr);
          if (!pack)
               return;
          assert (pack->acc_length >= IP_MIN_HDR_SIZE);
          ip_hdr= (ip_hdr_t *)ptr2acc_data(pack);
          assert (!(ntohs(ip_hdr->ih_flags_fragoff) &
               (IH_FRAGOFF_MASK|IH_MORE_FRAGS)));
     }
     size= ntohs(ip_hdr->ih_length);
     if (size > bf_bufsize(pack))
     {
          /* Should discard packet */
```

```
        assert(0);
        bf_afree(pack); pack= NULL;
        return;
    }

    exp_time= get_time() + (ip_hdr->ih_ttl+1) * HZ;

    if (ip_hdr->ih_dst == ip_port->ip_ipaddr)
        ip_pack_stat= NWIO_EN_LOC;
    else
        /*
        !my
        in real here the packets arrived by oroute or iroute tables
        that lead to the local interface
        assume NWIO_EN_ROUTE users
        */
        ip_pack_stat= NWIO_EN_BROAD;

    proto= ip_hdr->ih_proto;
    hash= proto & (IP_PROTO_HASH_NR-1);

    first_fd= NULL;
    for (i= 0; i<2; i++)
    {
        share_fd= NULL;

        ip_fd= (i == 0) ? ip_port->ip_proto_any :
            ip_port->ip_proto[hash];
        for (; ip_fd; ip_fd= ip_fd->if_proto_next)
```

```
        {
            if (i && ip_fd->if_ipopt.nwio_proto != proto)
                continue;
            if (!(ip_fd->if_ipopt.nwio_flags & ip_pack_stat))
                continue;
            if ((ip_fd->if_ipopt.nwio_flags & NWIO_REMSPEC) &&
                ip_hdr->ih_src != ip_fd->if_ipopt.nwio_rem)
            {
                continue;
            }
            if ((ip_fd->if_ipopt.nwio_flags & NWIO_ACC_MASK) ==
                NWIO_SHARED)
            {
                if (!share_fd)
                {
                    share_fd= ip_fd;
                    continue;
                }
                if (!ip_fd->if_rdbuf_head)
                    share_fd= ip_fd;
                continue;
            }
            if (!first_fd)
            {
                first_fd= ip_fd;
                continue;
            }
            pack->acc_linkC++;
            ip_packet2user(ip_fd, pack, exp_time, size);
```

```
        }
        if (share_fd)
        {
            pack->acc_linkC++;
            ip_packet2user(share_fd, pack, exp_time, size);
        }
    }
    if (first_fd)
    {
        if (first_fd->if_put_pkt &&
            (first_fd->if_flags & IFF_READ_IP) &&
            !(first_fd->if_ipopt.nwio_flags & NWIO_RWDATONLY))
        {
            (*first_fd->if_put_pkt)(first_fd->if_srfd, pack,
                size);
        }
        else
            ip_packet2user(first_fd, pack, exp_time, size);
    }
    else
    {
        if (ip_pack_stat == NWIO_EN_LOC)
        {
            DBLOCK(0x01,
            printf("ip_port_arrive: dropping packet for proto %d\n",
                proto));
        }
        else
```

```
          {
               DBLOCK(0x20, printf("dropping packet for proto %d\n",
                    proto));
          }
          bf_afree(pack);
     }
}

PUBLIC void ip_arrived(ip_port, pack)
ip_port_t *ip_port;
acc_t *pack;
{
     ip_hdr_t *ip_hdr;
     ipaddr_t dest;
     int ip_frag_len, ip_hdr_len, highbyte;
     size_t pack_size;
     acc_t *tmp_pack, *hdr_pack;
     ev_arg_t ev_arg;

     /*!my NAT*/
     pack = ip_nat_read(ip_port, pack);
     if(!pack){ return; }

     /**/
     pack_size= bf_bufsize(pack);

     if (pack_size < IP_MIN_HDR_SIZE)
     {
          DBLOCK(1, printf("wrong acc_length\n"));
```

```
            bf_afree(pack);
            return;
        }
        pack= bf_align(pack, IP_MIN_HDR_SIZE, 4);
        pack= bf_packIffLess(pack, IP_MIN_HDR_SIZE);
assert (pack->acc_length >= IP_MIN_HDR_SIZE);

        ip_hdr= (ip_hdr_t *)ptr2acc_data(pack);
        ip_hdr_len= (ip_hdr->ih_vers_ihl & IH_IHL_MASK) << 2;
        if (ip_hdr_len>IP_MIN_HDR_SIZE)
        {
            pack= bf_packIffLess(pack, ip_hdr_len);
            ip_hdr= (ip_hdr_t *)ptr2acc_data(pack);
        }
        ip_frag_len= ntohs(ip_hdr->ih_length);
        if (ip_frag_len != pack_size)
        {
            if (pack_size < ip_frag_len)
            {
                /* Sent ICMP? */
                DBLOCK(1, printf("wrong acc_length\n"));
                bf_afree(pack);
                return;
            }
            assert(ip_frag_len<pack_size);
            tmp_pack= pack;
            pack= bf_cut(tmp_pack, 0, ip_frag_len);
            bf_afree(tmp_pack);
            pack_size= ip_frag_len;
```

```
    }

    if (!ip_frag_chk(pack))
    {
         DBLOCK(1, printf("fragment not allright\n"));
         bf_afree(pack);
         return;
    }

    /* Decide about local delivery or routing. Local delivery can happen
     * when the destination is the local ip address, or one of the
     * broadcast addresses and the packet happens to be delivered
     * point-to-point.
     */

    /*
    //!my
    and local delivery can happen when oroute table lead route of dst address to the
interface
    if fact we need to know is the packet phisically arrived or it arrived by loopback event
    in the case we no need any checkup except delivery process
    ip_port_arrive does the delivery?
    */
    /*printf("read here: %u\n", );*/

    dest= ip_hdr->ih_dst;

    if (dest == ip_port->ip_ipaddr)
    {
```

```
        ip_port_arrive (ip_port, pack, ip_hdr);
        return;
    }
    if (broadcast_dst(ip_port, dest))
    {
        ip_port_arrive (ip_port, pack, ip_hdr);
        return;
    }

    if (pack->acc_linkC != 1 || pack->acc_buffer->buf_linkC != 1)
    {
        /* Get a private copy of the IP header */
        hdr_pack= bf_memreq(ip_hdr_len);
        memcpy(ptr2acc_data(hdr_pack), ip_hdr, ip_hdr_len);
        pack= bf_delhead(pack, ip_hdr_len);
        hdr_pack->acc_next= pack;
        pack= hdr_pack; hdr_pack= NULL;
        ip_hdr= (ip_hdr_t *)ptr2acc_data(pack);
    }
    assert(pack->acc_linkC == 1);
    assert(pack->acc_buffer->buf_linkC == 1);

    /* Try to decrement the ttl field with one. */
    if (ip_hdr->ih_ttl < 2)
    {
        icmp_snd_time_exceeded(ip_port->ip_port, pack,
            ICMP_TTL_EXC);
        return;
    }
```

```
    ip_hdr->ih_ttl--;
    ip_hdr_chksum(ip_hdr, ip_hdr_len);

    /* Avoid routing to bad destinations. */
    highbyte= ntohl(dest) >> 24;
    if (highbyte == 0 || highbyte == 127 ||
         (highbyte == 169 && (((ntohl(dest) >> 16) & 0xff) == 254)) ||
         highbyte >= 0xe0)
    {
         /* Bogus destination address */
         bf_afree(pack);
         return;
    }

    /* Further processing from an event handler */
    if (pack->acc_linkC != 1)
    {
         tmp_pack= bf_dupacc(pack);
         bf_afree(pack);
         pack= tmp_pack;
         tmp_pack= NULL;
    }
    pack->acc_ext_link= NULL;
    if (ip_port->ip_routeq_head)
    {
         ip_port->ip_routeq_tail->acc_ext_link= pack;
         ip_port->ip_routeq_tail= pack;
         return;
    }
```

```
    ip_port->ip_routeq_head= pack;
    ip_port->ip_routeq_tail= pack;
    ev_arg.ev_ptr= ip_port;
    ev_enqueue(&ip_port->ip_routeq_event, route_packets, ev_arg);
}

PUBLIC void ip_arrived_broadcast(ip_port, pack)
ip_port_t *ip_port;
acc_t *pack;
{
    ip_hdr_t *ip_hdr;
    int ip_frag_len, ip_hdr_len;
    size_t pack_size;
    acc_t *tmp_pack;

    pack_size= bf_bufsize(pack);

    if (pack_size < IP_MIN_HDR_SIZE)
    {
        DBLOCK(1, printf("wrong acc_length\n"));
        bf_afree(pack);
        return;
    }
    pack= bf_align(pack, IP_MIN_HDR_SIZE, 4);
    pack= bf_packIffLess(pack, IP_MIN_HDR_SIZE);
assert (pack->acc_length >= IP_MIN_HDR_SIZE);

    ip_hdr= (ip_hdr_t *)ptr2acc_data(pack);
```

```
    DIFBLOCK(0x20, (ip_hdr->ih_dst & HTONL(0xf0000000)) == HTONL(0xe0000000),
         printf("got multicast packet\n"));

    ip_hdr_len= (ip_hdr->ih_vers_ihl & IH_IHL_MASK) << 2;
    if (ip_hdr_len>IP_MIN_HDR_SIZE)
    {
         pack= bf_align(pack, IP_MIN_HDR_SIZE, 4);
         pack= bf_packIffLess(pack, ip_hdr_len);
         ip_hdr= (ip_hdr_t *)ptr2acc_data(pack);
    }
    ip_frag_len= ntohs(ip_hdr->ih_length);
    if (ip_frag_len<pack_size)
    {
         tmp_pack= pack;
         pack= bf_cut(tmp_pack, 0, ip_frag_len);
         bf_afree(tmp_pack);
    }

    if (!ip_frag_chk(pack))
    {
         DBLOCK(1, printf("fragment not allright\n"));
         bf_afree(pack);
         return;
    }

    if (!broadcast_dst(ip_port, ip_hdr->ih_dst))
    {
#if 0
```

```
            printf(
            "ip[%d]: broadcast packet for ip-nonbroadcast addr, src=",
                ip_port->ip_port);
            writeIpAddr(ip_hdr->ih_src);
            printf(" dst=");
            writeIpAddr(ip_hdr->ih_dst);
            printf("\n");
#endif
            bf_afree(pack);
            return;
        }

        ip_port_arrive (ip_port, pack, ip_hdr);
}

PRIVATE void route_packets(ev, ev_arg)
event_t *ev;
ev_arg_t ev_arg;
{
        ip_port_t *ip_port;
        ipaddr_t dest;
        acc_t *pack;
        iroute_t *iroute;
        ip_port_t *next_port;
        int r, type;
        ip_hdr_t *ip_hdr;
        size_t req_mtu;

        ip_port= ev_arg.ev_ptr;
```

```
    assert(&ip_port->ip_routeq_event == ev);

    while (pack= ip_port->ip_routeq_head, pack != NULL)
    {
         ip_port->ip_routeq_head= pack->acc_ext_link;

         ip_hdr= (ip_hdr_t *)ptr2acc_data(pack);
         dest= ip_hdr->ih_dst;

         iroute= iroute_frag(ip_port->ip_port, dest);
         if (iroute == NULL || iroute->irt_dist == IRTD_UNREACHABLE)
         {
              /* Also unreachable */
              /* Finding out if we send a network unreachable is too
               * much trouble.
               */
              if (iroute == NULL)
              {
                   printf("ip[%d]: no route to ",
                        ip_port-ip_port_table);
                   writeIpAddr(dest);
                   printf("\n");
              }
              icmp_snd_unreachable(ip_port->ip_port, pack,
                   ICMP_HOST_UNRCH);
              continue;
         }
         next_port= &ip_port_table[iroute->irt_port];
```

```
        if (ip_hdr->ih_flags_fragoff & HTONS(IH_DONT_FRAG))
        {
             req_mtu= bf_bufsize(pack);
             if (req_mtu > next_port->ip_mtu ||
                  (iroute->irt_mtu && req_mtu>iroute->irt_mtu))
             {
                  icmp_snd_mtu(ip_port->ip_port, pack,
                       next_port->ip_mtu);
                  continue;
             }
        }

        if (next_port != ip_port)
        {
             if (iroute->irt_gateway != 0)
             {
                  /* Just send the packet to the next gateway */
                  pack->acc_linkC++; /* Extra ref for ICMP */
                  r= next_port->ip_dev_send(next_port,
                       iroute->irt_gateway,
                       pack, IP_LT_NORMAL);
                  if (r == EDSTNOTRCH)
                  {
                       printf("ip[%d]: gw ",
                            ip_port-ip_port_table);
                       writeIpAddr(iroute->irt_gateway);
                       printf(" on ip[%d] is down for dest ",
                            next_port-ip_port_table);
                       writeIpAddr(dest);
```

```
                        printf("\n");
                        icmp_snd_unreachable(next_port-
                            ip_port_table, pack,
                            ICMP_HOST_UNRCH);
                        pack= NULL;
                    }
                    else
                    {
                        assert(r == 0);
                        bf_afree(pack); pack= NULL;
                    }
                    continue;
                }
                /* The packet is for the attached network. Special
                 * addresses are the ip address of the interface and
                 * net.0 if no IP_42BSD_BCAST.
                 */
                if (dest == next_port->ip_ipaddr)
                {
                    ip_port_arrive (next_port, pack, ip_hdr);
                    continue;
                }
                if (dest == iroute->irt_dest)
                {
                    /* Never forward obsolete directed broadcasts */
#if IP_42BSD_BCAST && 0
                    type= IP_LT_BROADCAST;
#else
                    /* Bogus destination address */
```

```
                    DBLOCK(1, printf(
                "ip[%d]: dropping old-fashioned directed broadcast ",
                                ip_port-ip_port_table);
                            writeIpAddr(dest);
                            printf("\n"););
                        icmp_snd_unreachable(next_port-ip_port_table,
                            pack, ICMP_HOST_UNRCH);
                        continue;
#endif
                }
                else if (dest == (iroute->irt_dest |
                    ~iroute->irt_subnetmask))
                {
                    if (!ip_forward_directed_bcast)
                    {
                        /* Do not forward directed broadcasts */
                        DBLOCK(1, printf(
                        "ip[%d]: dropping directed broadcast ",
                                    ip_port-ip_port_table);
                            writeIpAddr(dest);
                            printf("\n"););
                        icmp_snd_unreachable(next_port-
                            ip_port_table, pack,
                            ICMP_HOST_UNRCH);
                        continue;
                    }
                    else
                        type= IP_LT_BROADCAST;
                }
```

```
            else
                type= IP_LT_NORMAL;

            /* Just send the packet to it's destination */
            pack->acc_linkC++; /* Extra ref for ICMP */
            r= next_port->ip_dev_send(next_port, dest, pack, type);
            if (r == EDSTNOTRCH)
            {
                DBLOCK(1, printf("ip[%d]: next hop ",
                    ip_port-ip_port_table);
                    writeIpAddr(dest);
                    printf(" on ip[%d] is down\n",
                    next_port-ip_port_table););
                icmp_snd_unreachable(next_port-ip_port_table,
                    pack, ICMP_HOST_UNRCH);
                pack= NULL;
            }
            else
            {
                assert(r == 0 || (printf("r = %d\n", r), 0));
                bf_afree(pack); pack= NULL;
            }
            continue;
        }

        /* Now we know that the packet should be routed over the same
         * network as it came from. If there is a next hop gateway,
         * we can send the packet to that gateway and send a redirect
         * ICMP to the sender if the sender is on the attached
```

```
         * network. If there is no gateway complain.
         */
        if (iroute->irt_gateway == 0)
        {
            printf("ip_arrived: packet should not be here, src=");
            writeIpAddr(ip_hdr->ih_src);
            printf(" dst=");
            writeIpAddr(ip_hdr->ih_dst);
            printf("\n");
            bf_afree(pack);
            continue;
        }
        if (((ip_hdr->ih_src ^ ip_port->ip_ipaddr) &
            ip_port->ip_subnetmask) == 0)
        {
            /* Finding out if we can send a network redirect
             * instead of a host redirect is too much trouble.
             */
            pack->acc_linkC++;
            icmp_snd_redirect(ip_port->ip_port, pack,
                ICMP_REDIRECT_HOST, iroute->irt_gateway);
        }
        else
        {
            printf("ip_arrived: packet is wrongly routed, src=");
            writeIpAddr(ip_hdr->ih_src);
            printf(" dst=");
            writeIpAddr(ip_hdr->ih_dst);
            printf("\n");
```

```
                printf("in port %d, output %d, dest net ",
                    ip_port->ip_port,
                    iroute->irt_port);
                writeIpAddr(iroute->irt_dest);
                printf("/");
                writeIpAddr(iroute->irt_subnetmask);
                printf(" next hop ");
                writeIpAddr(iroute->irt_gateway);
                printf("\n");
                bf_afree(pack);
                continue;
            }
            /* No code for unreachable ICMPs here. The sender should
             * process the ICMP redirect and figure it out.
             */
            ip_port->ip_dev_send(ip_port, iroute->irt_gateway, pack,
                IP_LT_NORMAL);
        }
}

PRIVATE int broadcast_dst(ip_port, dest)
ip_port_t *ip_port;
ipaddr_t dest;
{
        ipaddr_t my_ipaddr, netmask, classmask;

        /* Treat class D (multicast) address as broadcasts. */
        if ((dest & HTONL(0xF0000000)) == HTONL(0xE0000000))
        {
```

```
        return 1;
    }

    /* Accept without complaint if netmask not yet configured. */
    if (!(ip_port->ip_flags & IPF_NETMASKSET))
    {
        return 1;
    }
    /* Two possibilities, 0 (iff IP_42BSD_BCAST) and -1 */
    if (dest == HTONL((ipaddr_t)-1))
        return 1;
#if IP_42BSD_BCAST
    if (dest == HTONL((ipaddr_t)0))
        return 1;
#endif
    netmask= ip_port->ip_subnetmask;
    my_ipaddr= ip_port->ip_ipaddr;

    if (((my_ipaddr ^ dest) & netmask) != 0)
    {
        classmask= ip_port->ip_classfulmask;

        /* Not a subnet broadcast, maybe a classful broadcast */
        if (((my_ipaddr ^ dest) & classmask) != 0)
        {
            return 0;
        }
        /* Two possibilities, net.0 (iff IP_42BSD_BCAST) and net.-1 */
        if ((dest & ~classmask) == ~classmask)
```

```
        {
            return 1;
        }
#if IP_42BSD_BCAST
        if ((dest & ~classmask) == 0)
            return 1;
#endif
        return 0;
    }

    if (!(ip_port->ip_flags & IPF_SUBNET_BCAST))
        return 0;/* No subnet broadcasts on this network */

    /* Two possibilities, subnet.0 (iff IP_42BSD_BCAST) and subnet.-1 */
    if ((dest & ~netmask) == ~netmask)
        return 1;
#if IP_42BSD_BCAST
    if ((dest & ~netmask) == 0)
        return 1;
#endif
    return 0;
}

/*
!my
*/
PUBLIC void ip_loopb_arrived(ip_port, pack)
ip_port_t *ip_port;
acc_t *pack;
```

```
{
     ip_hdr_t *ip_hdr;
     int ip_frag_len, ip_hdr_len;
     size_t pack_size;

     pack_size= bf_bufsize(pack);

     assert (pack_size >= IP_MIN_HDR_SIZE);
     pack= bf_align(pack, IP_MIN_HDR_SIZE, 4);
     pack= bf_packIffLess(pack, IP_MIN_HDR_SIZE);
     assert (pack->acc_length >= IP_MIN_HDR_SIZE);

     ip_hdr= (ip_hdr_t *)ptr2acc_data(pack);
     ip_hdr_len= (ip_hdr->ih_vers_ihl & IH_IHL_MASK) << 2;
     if (ip_hdr_len>IP_MIN_HDR_SIZE)
     {
          pack= bf_packIffLess(pack, ip_hdr_len);
          ip_hdr= (ip_hdr_t *)ptr2acc_data(pack);
     }
     ip_frag_len= ntohs(ip_hdr->ih_length);
     assert(ip_frag_len == pack_size);

     /* here local delivery only */
     ip_port_arrive (ip_port, pack, ip_hdr);
}

void ip_process_loopb(ev, arg)
event_t *ev;
ev_arg_t arg;
```

```
{
	ip_port_t *ip_port;
	acc_t *pack;

	ip_port= arg.ev_ptr;
	assert(ev == &ip_port->ip_loopb_event);

	while(pack= ip_port->ip_loopb_head, pack != NULL)
	{
		ip_port->ip_loopb_head= pack->acc_ext_link;
		/*
		ip_arrived(ip_port, pack);
		//!my
		*/
		ip_loopb_arrived(ip_port, pack);
	}
}

/*
 * $PchId: ip_read.c,v 1.33 2005/06/28 14:18:50 philip Exp $
 */
```

.

-) Файл src/servers/inet/generic/ip.c

Изменения в этот файл:
строка 24:

```
#include "sr.h"

/*!my ip_nat.c */
```

```
EXTERN void ip_nat_init(void);
```

строка 62:

```
    assert (BUF_S >= sizeof(nwio_route_t));

    /*!my ip_nat */
    ip_nat_init();

    for (i=0, ip_ass= ip_ass_table; i<IP_ASS_NR; i++, ip_ass++)
```

.

-) Файл src/servers/inet/generic/ipr.c

В файле ipr.c меняем функцию oroute_frag, добавляя возвращаемый признак "шлюза по умолчанию", если это не сработает, шлюз по умолчанию можно будет задавать статически и обрабатывать его после всех маршрутов (в ip_write.c), когда ни один маршрут не подошел.

строка 190:

```
/*!my*/
PUBLIC int oroute_frag(port_nr, dest, ttl, msgsize, nexthop, is_def_gw)
int port_nr;
ipaddr_t dest;
int ttl;
size_t msgsize;
ipaddr_t *nexthop;
char *is_def_gw;
{
    oroute_t *oroute;

    oroute= oroute_find_ent(port_nr, dest);
    if (!oroute || oroute->ort_dist > ttl)
        return EDSTNOTRCH;
```

```
    if (msgsize && oroute->ort_mtu &&
        oroute->ort_mtu < msgsize)
    {
        return EPACKSIZE;
    }

    *nexthop= oroute->ort_gateway;
    /*!my*/
    *is_def_gw = !(oroute->ort_subnetmask);
    #if DVI_OR > 1
    printf("oroute dst: "); writeIpAddr(oroute->ort_dest);
    printf(" mask: "); writeIpAddr(oroute->ort_subnetmask);
    printf( "\n" );
    #endif

    return NW_OK;
}
```

.

-) Файл src/servers/inet/generic/ipr.h

Меняем объявление этой функции в файле ipr.h
строка 60:

```
/*!my*/
int oroute_frag ARGS(( int port_nr, ipaddr_t dest, int ttl, size_t msgsize,
                                ipaddr_t *nexthop, char *is_def_gw ));
void ipr_init ARGS(( void ));
```

.

-) Файл src/servers/inet/generic/ip_int.h

строка 182:

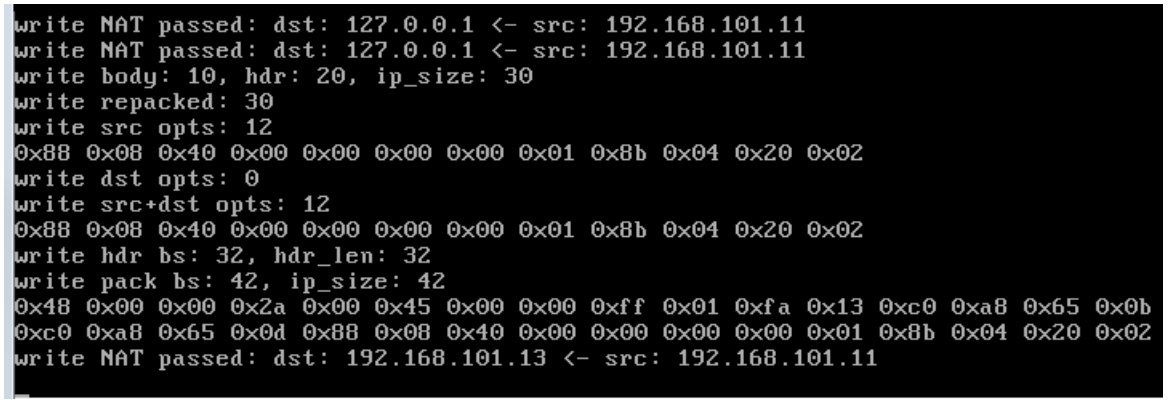
```
extern ip_ass_t ip_ass_table[IP_ASS_NR];

/*!my */
#define NWIO_DEFAULT    (NWIO_EN_LOC | NWIO_EN_BROAD | NWIO_REMANY | \
     NWIO_RWDATALL | NWIO_HDR_O_SPEC \
     | NWIO_DI_ROUTE | NWIO_SRCAUTO | NWIO_IDAUTO | NWIO_HDRAUTO \
     )

#endif /* INET_IP_INT_H */
```

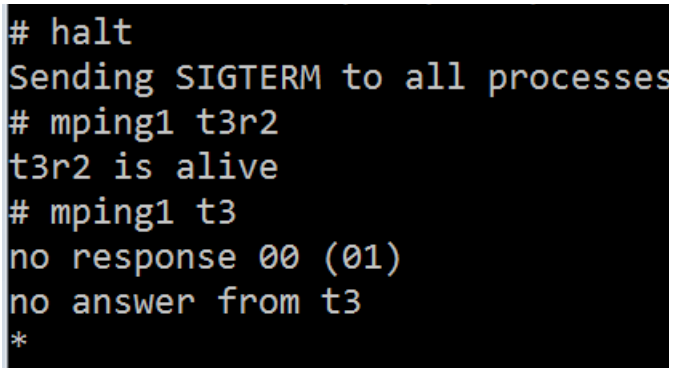
.

Результаты работы третьего примера.

В целом IPv4r2 NAT это была видимо самая простая в мире программа по составлению одного буфера из двух частей, которая давалась с таким трудом, так что мне приходилось отлаживать практически каждую строчку, поскольку ошибки возникали так много и так часто и так неожиданно, что я прямо занимался их поиском.

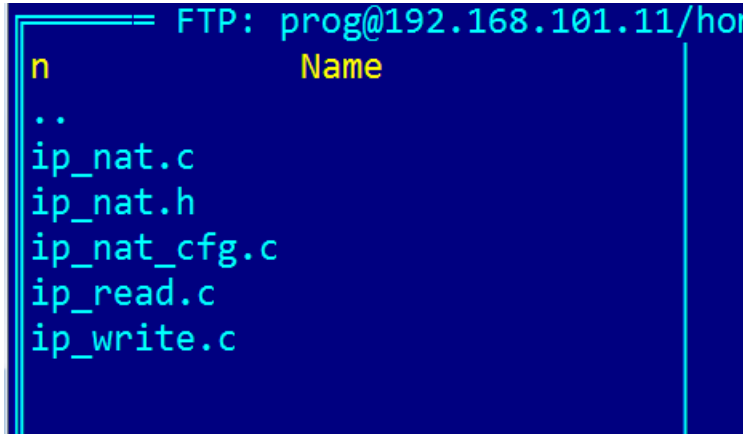
Силы зла в это время властвовали над болотами безраздельно...

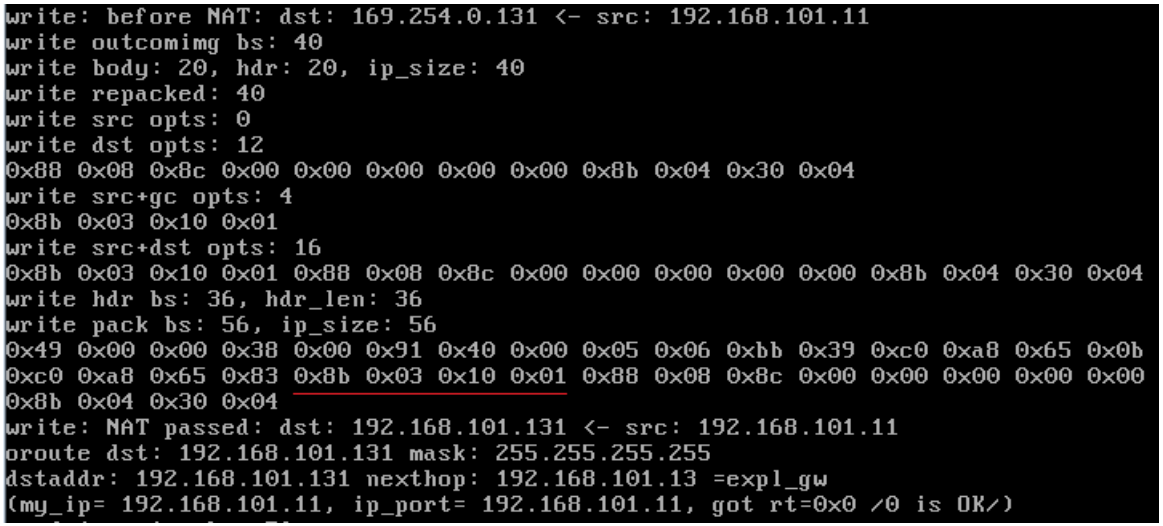




работа ping IPv4r2
Слева: работа NAT IPv4r2

.





Доступ по R2 ftp к исходным файлам IPv4r2
Справа: Команда шлюза GC в заголовке

Беседы за кружкой чая:

- опция 88 08 будет работать;
- тогда включайте ее;
- что то опция 88 08 не работает;
- ха, ну так естественно, оказалось что вы не знаете всех тонкостей протоколов IPv4;
- вы с причин неработоспособности опции 88 08 переходите на мои личные качества (рассматриваются мои познания в протоколах IPv4, мое предварительное мнение об этих своих познаниях) и меняете тему разговора (я и опция 88 08 не одно и то же), а это лжесвидетельство двух видов;
- хм, хорошо, поясню, ваша опция 88 08 не работает, потому что вы не знаете всех тонкостей протоколов IPv4;
- вы с причин неработоспособности опции 88 08 переходите на мои личные качества (рассматриваются мои познания в протоколах IPv4) и меняете тему разговора (я и опция 88 08 не одно и то же), а это лжесвидетельство двух видов, в случае таких причин нужно как минимум рассматривать "работу опции 88 08 с протоколами IPv4"; во-вторых, если мне самому попытаться восстановить ваше заявление в понятную форму, варианты по смыслу, который мог бы быть заложен в исходной форме:
    - если "я включаю опцию 88 08, но сама по себе эта опция не может работать", то эта опция работать может, работу опции 88 08 я представляю себе совершенно отчетливо, тем более я сам ее сделал;
    - если "я включаю опцию 88 08, но по неизвестной причине эта опция с протоколами IPv4 не работает (причина скрыта тоже из-за незнания протоколов IPv4)", другими словами "опция 88 08 не работает по неизвестной причине, зависящей

от протоколов IPv4", то другие протоколы специально разделены по сетевым уровням, чтобы их можно было бы не знать, т.е. работа опции 88 08 не должна зависеть от любого другого протокола любого другого сетевого уровня, поскольку интерфейс IPv4 с введением опции 88 08 не изменяется;

- ага, так значит вы так самоуверенны в том, что хорошо знаете все тонкости протоколов IPv4!

```
read incomimg bs: 50
read opt_len: 12 ip_size: 42
0x88 0x08 0x40 0x00 0x00 0x00 0x00 0x01 0x8b 0x04 0x20 0x02
read before delhead: 50
read body: 18, hdr: 32, ip_size: 42
read dst opt found: B45: 0, U12: 0
read wrong own IPv4r2: 192.168.101.13
0x88 0x08 0x40 0x00 0x00 0x00 0x00 0x01 0x8b 0x04 0x20 0x02
```

IPv4 minix ftp клиент с поддержкой R2 .

Список измененных файлов

- src/commands/ftp/file.c
- src/commands/ftp/ftp.c
- src/commands/ftp/ftp.h
- src/commands/ftp/net.c
- src/commands/ftp/other.c
- src/commands/ftp/other.h

-)Файл src/commands/ftp/file.c

Строка 421:

```
char *file;

    /*!my*/
    printf("FTP mode:\n");
    printf("    PASSIVE: %s\n", (passive? "ON": "OFF") );
    printf("    R2:      %s\n", (is_R2? "ON": "OFF") );
    if(is_R2 && !is_R2_server ){
    printf("    R2:      %s\n", "server does not suport R2" );}
```

```
   if(cmdargc < 2)
```

-)Файл src/commands/ftp/ftp.c

Строка 26:

```
#include "net.h"

/*!my*/
char is_R2;
char is_R2_server;
ipaddr_t remote_ip;

FILE *fpcommin;
```

Строка 198:

```
   /*!my*/
   readline("Press [q+] ENTER to continue... ", junk, sizeof(junk));
   switch(junk[0]){ case 'q': case 'Q': return 0; }
```

Строка 212:

```
   printf("quote         Send raw ftp command to remote host\n");
   /*!my*/
   printf("r2            Toggle R2 mode\n");
   printf("reget         Restart a partial file retrieve from remote host\n");
```

Строка 267:

```
    "quote",        DOquote,
   /*!my*/
    "r2",         DOr2,
```

-)Файл src/commands/ftp/ftp.h

Строка 10:

```
#include <net/gen/in.h>

/*!my R2 mode*/
extern char is_R2;
extern char is_R2_server;
extern ipaddr_t remote_ip;

extern FILE *fpcommin;
```

-)Файл src/commands/ftp/net.c

Строка 102:

```
   /*!my*/
   remote_ip= hostip;
   /* use R2 mode in the case */
   #if 0
   /* This HACK allows the server to establish data connections correctly */
   /* when using the loopback device to talk to ourselves */
     if(hostip == inet_addr("127.0.0.1"))
     hostip = myip;
   #endif
```

Строка 242:

```
   /*!my*/
   if(is_R2){
   if(is_R2_server){
     s = DOcommand("R2", "");
     if(s != 327){
```

```
        close(ftpdata_fd);
        return s;
     }
   }}
```

Строка 278:

```
    /*!my*/
     /*
     ripaddr = ntohl(ripaddr);
     rport = ntohs(rport);
    */
     ripaddr = htonl(ripaddr);
     rport = htons(rport);
   }

   /*!my*/
   if(is_R2)ripaddr = remote_ip;
```

-)Файл src/commands/ftp/other.c

Строка 21:

```
void FTPinit()
{
   /*!my*/
   is_R2= 0;
   is_R2_server= 0;
```

Строка 119:

```
/*!my*/
int DOr2()
{
```

```
   is_R2= 1 - is_R2;
   is_R2_server= is_R2;

   printf("R2 mode is now %s\n", (is_R2 ? "ON" : "OFF"));
   if(!is_R2)return 0;

   /*check R2 0*/
   if( cmdargc > 1 ){
   if( cmdargv[1][0] == '0' ){
     is_R2_server= 0;
     printf("Set 'server does not support R2'\n");
   }}

   return(0);
}
```

-)Файл src/commands/ftp/other.h

Строка 14:

```
_PROTOTYPE(int DOpassive, (void));
/*!my*/
_PROTOTYPE(int DOr2, (void));
_PROTOTYPE(int DOsyst, (void));
```

IPv4 minix ftpd сервер с поддержкой R2 .

Список измененных файлов

- src/commands/ftpd200/ftpd.c
- src/commands/ftpd200/ftpd.h
- src/commands/ftpd200/net.c

- src/commands/ftpd200/net.h

-)Файл src/commands/ftpd/ftpd.c

Строка 42:

```
#include "net.h"

/*!my*/
char is_R2;
ipaddr_t remote_ip;
```

Строка 112:

```
     "QUIT", doQUIT,
    /*!my*/
     "R2",   doR2,
```

-)Файл src/commands/ftpd/ftpd.h

Строка 9:

```
/*!my R2 mode*/
extern char is_R2;
extern ipaddr_t remote_ip;

#define  GOOD 0
```

-)Файл src/commands/ftpd/net.c

Строка 59:

```
int retry;
/*!my*/
char cur_is_R2;
```

```
    /*!my*/
    cur_is_R2= is_R2;
    is_R2= 0;
```

Строка 194:

```
int i;
/*!my*/
char cur_is_R2;

    /*!my*/
    cur_is_R2= is_R2;
    is_R2= 0;
```

Строка 226:

```
  /*!my*/
  if(!cur_is_R2){
  /*avoid DDoS*/
  if(dataaddr != rmtipaddr) {
    printf(msg501);
    return(GOOD);
  }}

  /*!my*/
  /*printf("200 Port command okay.\r\n");*/
  printf("200 Port (%u,%u,%u,%u,%u,%u).\r\n",
    hibyte(hiword(ntohl(dataaddr))), lobyte(hiword(ntohl(dataaddr))),
    hibyte(loword(ntohl(dataaddr))), lobyte(loword(ntohl(dataaddr))),
    hibyte(ntohs(dataport)), lobyte(ntohs(dataport)));
```

```
   /*!my*/
   if(cur_is_R2)dataaddr= remote_ip;

   return(GOOD);
```

Строка 247:

```
/*!my*/
int doR2(buff)
char *buff;
{
   is_R2= 1;

   printf("327 R2, data transfer uses the same IP.\r\n");
   return(GOOD);
}
```

Строка 451:

```
   /*!my*/
   remote_ip= rmtipaddr;

   /* Look up the host name of the remote host. */
```

-)Файл src/commands/ftpd/net.h

Строка 11:

```
/*!my*/
_PROTOTYPE(int doR2, (char *buff));
```

Пример реализации ftp R2 на minix. На рисунке слева IPv4 ftp протокол успешно победил адрес 127.0.0.1 с помощью режима "R2 0" (по большому счету весь протокол IPv4r2 мало отличается от использования подстановочного адреса 127.0.0.1), на рисунке справа режим R2 поддержан и клиентом и сервером.

Победа IPv4 ftp над адресом 127.0.0.1 в режиме R2
Слева: режим "R2 0", только R2 клиент
Справа: режим "R2", R2 клиент и сервер

```
ftp>R2 0
R2 mode is now ON
Set 'server does not support R2'
ftp>passive
Passive mode is now ON
ftp>ls
227 Entering Passive Mode (192,168,101,11,128,7)
125 File NLST okay.  Opening data connection.
.ashrc
.ellepro.b1
.ellepro.e
.exrc
.profile
my
226 Transfer finished successfully. 0.05 KB/s
ftp>status
FTP mode:
    PASSIVE: ON
    R2:      ON
    R2:      server does not suport R2
211-mx1(192.168.101.11:21) FTP server status:
    Version 2.00  Tue, 01 Mar 2016 21:38:46 GMT
    Connected to 127.0.0.1:32771
    Logged in prog
    MODE: Stream
    TYPE: Ascii
211 End of status
ftp>
```

```
ftp>R2
R2 mode is now ON
ftp>ls
327 R2, data transfer uses the same IP.
200 Port command okay.
150 File NLST okay.  Opening data connection.
.ashrc
.ellepro.b1
.ellepro.e
.exrc
.profile
my
226 Transfer finished successfully. 0.05 KB/s
ftp>status
FTP mode:
    PASSIVE: OFF
    R2:      ON
211-mx1(192.168.101.11:21) FTP server status:
    Version 2.00  Tue, 01 Mar 2016 22:35:06 GMT
    Connected to 127.0.0.1:32772
    Logged in prog
    MODE: Stream
    TYPE: Ascii
211 End of status
ftp>
```

Улучшенная IPv4 утилита mping.

В minix утилита ping довольно скромная, поскольку она нужна часто, то это вариант немного улучшенного ping, которая решает интересные для отладки IPv4r2 сети задачи, файл mping.c

```
/*
ping.c
*/

#include <sys/types.h>
#include <errno.h>
#include <signal.h>
#include <net/gen/netdb.h>
#include <sys/ioctl.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <net/gen/oneCsum.h>
#include <fcntl.h>
#include <net/gen/in.h>
#include <net/gen/inet.h>
#include <net/gen/ip_hdr.h>
#include <net/gen/icmp_hdr.h>
#include <net/gen/ip_io.h>

#include <string.h>

#define DEBUG 1

#define WRITE_SIZE 30
```

```
#define PING_REQUESTS 5

char buffer[16*1024];
char dev_name[80];
const char *pname= 0;

#if DEBUG
#define where() fprintf(stderr, "%s %d:", __FILE__, __LINE__);
#endif

#if __STDC__
#define PROTO(x,y) x y
#else
#define PROTO(x,y) X ()
#endif

PROTO (int main, (int argc, char *argv[]) );
static PROTO (void sig_hand, (int signal) );
static PROTO (void writeIpAddr, (ipaddr_t addr) );
static PROTO (void usage, (const char *who, const char *msg) );

/**/
static void writeIpAddr(addr)
ipaddr_t addr;
{
#define addrInBytes ((u8_t *)&addr)

    fprintf(stdout, "%d.%d.%d.%d", addrInBytes[0], addrInBytes[1],
        addrInBytes[2], addrInBytes[3]);
```

```
#undef addrInBytes
}

static void usage(who, msg)
const char *who;
const char *msg;
{
     if(who||msg){
     fprintf(stderr, "error");
     if(who)fprintf(stderr, ": %s", who );
     if(msg)fprintf(stderr, ": %s", msg );
     fprintf(stderr, "\n" );
     }

     fprintf(stderr,
               "Usage: %s hostname [options]\n"
               "options:\n"
               "\t-d<ip dev number> (0,1,...)\n"
               "\t-n<ping requests> (1,2,...)\n"
               "\t-1 (break on first answer)\n"
               "\t-l<ping frame length> (%u,...)\n"
               "(ln 'mping' to 'mping1' implies -n1 -1 as default)"
               ,pname, WRITE_SIZE
               );
     exit(1);
}

static void sig_hand(signal)
int signal;
```

```
{
}

/**/
int  main(argc, argv)
int argc;
char *argv[];
{
     int fd, i;
     int result, result1;
     nwio_ipopt_t ipopt;
     ip_hdr_t *ip_hdr;
     int ihl;
     icmp_hdr_t *icmp_hdr;
     ipaddr_t dst_addr;
     struct hostent *hostent;

     nwio_ipconf_t ipconf;
     ipaddr_t      ip_own;

     unsigned length;
     unsigned ping_requests;
     unsigned ping_answers;
     unsigned tmp;
     char     is_open_line, is_msg_no_resp;
     char     is_one_answer;

     /**/
     length= WRITE_SIZE;
```

```
    ping_requests= PING_REQUESTS;
    ping_answers= 0;
    is_one_answer= 0;

    strcpy(dev_name, "/dev/ip");

    /*parse cmdline*/
    pname= argv[0];
    if(argc<2)usage("parameter missing","hostname");

    if( !strcmp(argv[0],"mping1") ){
         ping_requests= 1;
         is_one_answer= 1;
    }

    for( i=2; i<argc; ++i ){
    switch(argv[i][0]){
    case '-':
         if(argv[i][1]=='d'){
         tmp= strtoul(&argv[i][2],0,0);
         sprintf(
              dev_name+strlen(dev_name),
              "%u", tmp);
         break;}

         if(argv[i][1]=='n'){
         tmp= strtoul(&argv[i][2],0,0);
         if( !tmp  )
              usage("0 pings request",argv[i]);
```

```
        ping_requests= tmp;
        break;}

        if(argv[i][1]=='1'){
        is_one_answer= is_one_answer? 0: 1;
        break;}

        if(argv[i][1]=='l'){
        tmp= strtoul(&argv[i][2],0,0);
        if( tmp < (sizeof(icmp_hdr_t) + IP_MIN_HDR_SIZE) )
             usage("length too small",argv[i]);
        length= tmp;
        break;}

    default:
        usage("bad parameter",argv[i]);
    }}

    /**/
    hostent= gethostbyname(argv[1]);
    if (hostent)
        dst_addr= *(ipaddr_t *)(hostent->h_addr);
    else
    {
        dst_addr= inet_addr(argv[1]);
        if (dst_addr == -1) usage("unknown host",argv[1]);
    }

    /**/
```

```
    fd= open (dev_name, O_RDWR);
    if(fd<0){
         fprintf(stderr, "error: %s: %s\n",
                   "open", dev_name);
         perror("open");
         exit(1);
    }
    printf("device: %s\n", dev_name);

    /**/
    result= ioctl (fd, NWIOGIPCONF, &ipconf);
    if (result < 0){
         fprintf(stderr, "error: %s: %s\n",
                   "Unable to get IP configuration", dev_name);
         perror("NWIOGIPCONF");
         exit(1);
    }
    ip_own = ipconf.nwic_ipaddr;

    /**/
    ipopt.nwio_flags= NWIO_COPY | NWIO_PROTOSPEC | NWIO_REMSPEC;
    ipopt.nwio_proto= 1;
    ipopt.nwio_rem= dst_addr;

    result= ioctl (fd, NWIOSIPOPT, &ipopt);
    if (result<0){
         fprintf(stderr, "error: %s: %s\n",
                   "Unable to set IP options", dev_name);
         perror("NWIOSIPOPT");
```

```
        exit(1);
    }

    /**/
    is_open_line= 0;
    for (i= 0; i< ping_requests; i++)
    {
        ip_hdr= (ip_hdr_t *)buffer;
        memset(ip_hdr, 0, sizeof(ip_hdr));

        icmp_hdr= (icmp_hdr_t *)(buffer+20);
        icmp_hdr->ih_type= 8;
        icmp_hdr->ih_code= 0;
        icmp_hdr->ih_chksum= 0;
        icmp_hdr->ih_chksum= ~oneC_sum(0, (u16_t *)icmp_hdr, length-20);

        /**/
        result= write(fd, buffer, length);
        if (result<0){
            fprintf(stderr, "error: %s: %s\n",
                    "write", dev_name);
            perror("write");
            exit(1);
        }
        if (result != length)
        {
            fprintf(stderr, "error: %s: %u (expected %u): %s\n",
                    "incomplete write", result, length,
                    dev_name);
```

```
            perror("write");
            exit(1);
        }

        alarm(0);
        signal (SIGALRM, sig_hand);
        alarm(1);

        /**/
        result= read(fd, buffer, sizeof(buffer));
        if(result < 0){
            if(errno == EINTR){
                if(is_open_line && !is_msg_no_resp)printf("\n");
                fprintf(stdout, "\r%s no response: pass %02u of %02u|",
                          argv[1], i, ping_requests);
                fflush(stdout);
                is_open_line= 1;
                is_msg_no_resp= 1;
                continue;
            }

            if(is_open_line)printf("\n");
            fprintf(stderr, "error: %s: %s\n",
                      "read", dev_name);
            perror("read");
            continue;
        }
        #if 0
        if (result != length)
```

```
        {
            fprintf(stderr, "error: %s: %u (expected %u): %s\n",
                    "incomplete read", result, length,
                    dev_name);
            perror("read");
            continue;
        }
        #endif

        /**/
        if(
            ( ip_hdr->ih_src != ipopt.nwio_rem )
            ||( ip_hdr->ih_dst != ip_own )
            ||( ip_hdr->ih_proto != 1 )
            ||( icmp_hdr->ih_type != 0 )
            ){

            if(is_open_line)printf("\n");
            is_open_line= 0;

            printf("    %s: drop answer: ", argv[1]);
            if( ip_hdr->ih_proto == 1 )printf("ICMP 0x%x:", icmp_hdr->ih_type);
            else printf("prot 0x%x:", ip_hdr->ih_proto);
            printf(" "); writeIpAddr(ip_hdr->ih_src);
            printf(" -> "); writeIpAddr(ip_hdr->ih_dst);

            printf("\n    (expected ICMP 0x%x:", 0);
            printf(" "); writeIpAddr(ipopt.nwio_rem);
            printf(" -> ");  writeIpAddr(ip_own);
```

```
            printf(" )\n");
            continue;
        }

        if(is_open_line && is_msg_no_resp)printf("\n");

        #if 0
        printf("%s is alive\n", argv[1]);
        #else
        fprintf(stdout, "\r%s is alive:    pass %02u of %02u|",
                          argv[1], i, ping_requests);
        fflush(stdout);
        is_open_line= 1;
        is_msg_no_resp= 0;
        #endif

        ++ping_answers;
        if(is_one_answer)break;
    }

    /**/
    if(is_open_line)printf("\n");
    is_open_line= 0;

    if(!ping_answers){
    printf("no answer from %s\n", argv[1]);
    exit(1);
    }
```

```
    if(is_one_answer){
    printf("answer from %s (tried %u times of %u)\n",
           argv[1], i+1, ping_requests);
    exit(0);
    }

    printf("answers from %s (got %u times of %u)\n",
           argv[1], ping_answers, ping_requests);
    exit(0);
}

/**/
```

===
Конец текста