IPv4r2
расширение протокола IPv4
до версии (ревизии) два,
описание протокола от 12 декабря 2015 года,
поправки от 31 декабря 2015 года.

Полянин М.А.
31 декабря 2015.

Описания поправок.

1. В черновике IPv4r2 протокола от 12 декабря 2015 для протокола TCPr2 допущена ошибка - поле размер окна не расширено до 32 бит и не вставлено резервное поле 16 бит, но поскольку протоколы UDPr2/TCPr2 продвинулись в своем логическом развитии в плане формализации адресации, формат заголовков этих протоколов в текущем варианте полностью изменен в связи с появлением промежуточного протокола локальной адресации хоста HLAP.

2. Приведен рабочий пример реализации протокола IPv4r2.

Версии UDPr2/TCPr2 в стеке протоколов IPv4r2.

-) 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 пакета следующего протокола (в этот размер входит и 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.

-) 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.

Все остальные условия TCPr2 как в версии TCPr2 от 12 декабря 2015.

-) 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.

Все остальные условия UDPr2 как в версии UDPr2 от 12 декабря 2015.

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, я приведу пример двух программ для minix 3_1_0 (книжная версия), которые на этой minix машине:

- выполняют тестовую отправку и прием IPv4r2 пакетов IPv4r2 адресации в режиме совместимости с IPv4;
- занимаются NAT преобразованием IPv4r2 внешней адресации в локальные адреса IPv4 (это позволяет двум машинам обмениваться с помощью протокола IPv4r2 адресации без каких-либо изменений в IPv4 прикладных программах).

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 аналогичный 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.

В тесте регистрация IPv4r2 адреса в своей локальной сети делается вручную, т.е. вы должны узнать какие IPv4r2 адреса вам нужны (куда вы хотите подключиться) и вручную указать NAT программе создать для этих IPv4r2 адресов локальные IPv4 адреса на вашей машине.

Для полной интеграции IPv4r2:

- программы на стороне пользователя должны уметь регистрировать IPv4r2 адреса в своей локальной сети, обращаясь к

IPv4r2 серверу имен и получая локальный IPv4 адрес, актуальный для локальной машины. Такие удобства работы эквивалентны P-t-P протоколам типа торент или лучше.

- Еще потребуется чтобы провайдер зарегистрировал IPv4r2 адреса своих клиентов в своем DNSr2 сервисе.

Ну, а идеальной интеграции с IPv4r2 можно достигнуть если существующие приложения, использующие формат IPv4, смогут прямо понимать IPv4r2 адреса без их ретрансляции на локальной машине в локальные IPv4 адреса.

Тест IPv4r2 адресации в режиме совместимости с IPv4.

Для того чтобы в этом minix отправлять IPv4r2 пакеты надо внести изменения в ряд файлов.

-) 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
```

Адресом IPv4 из диапазона 169.254.0.0/16 для IPv4r2 трансляции мы оторвем кус от DHCP, поскольку твердо считаем, что назначать IPv4 адреса произвольно нет смысла:

- или это интерфейс в чужой общей сети где работает DHCP;
- или это интерфейс в вашей малой локальной сети, где адрес компьютера вам можно задать вручную и даже надписать его прямо на корпусе компьютера в виде четырех цифр, чтобы не забыть что это за компьютер;
- других вариантов не должно быть.

-) 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];
                  break;
```

Вот и все модификации.

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

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;
```

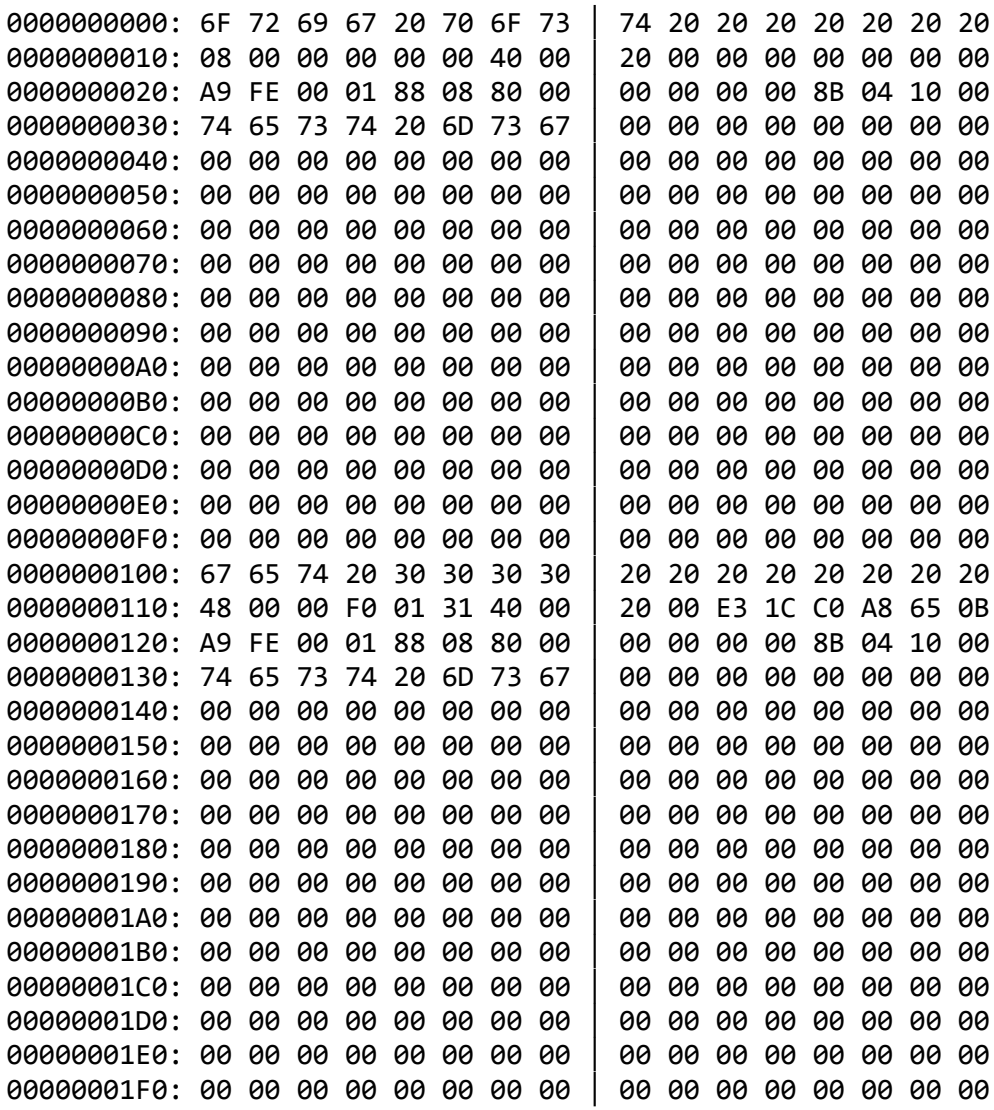
```
nwio_ipopt_t      ipopt;

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:



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 (следующий тест).

Тест NAT преобразования IPv4r2 адресации в локальные адреса IPv4.

Сделать на базе предыдущего опыта NAT отображение IPv4r2 на IPv4 не сильно сложно, но в двух словах об этом не расскажешь, здесь это будет многословно и непонятно, но выглядит это все в тестовом виде примерно так:
файл r2_nat.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 и достойного публикации, 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.

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, нам в общем то нет нужды, мы даже не знаем как работают их сети. Если внутри сетей провайдера им нужна IPv6 адресация, нам все равно.

===
Конец текста