Санкт-Петербургский Государственный Университет
Математико-Механический факультет
кафедра системного программирования

Сетевой стек реального времени

курсовая работа студента 445 группы Калмука А. И.

научный руководитель
аспирант кафедры системного программирования Абусалимов Э. Ш.

2013

Оглавление

Введение

В наши дни активно развиваются сетевые технологии. Уже трудно себе представить устройство без взаимодействия по сети. За это взаимодействие отвечает модуль операционной системы, часто называемый “сетевой стек”, функции которого могут варьироваться от системы к системе, но базовая задача – это предоставление возможности взаимодействия различных устройств по сети. Кроме того, на взаимодействие по сети, как правило, накладываются разные ограничения, такие как безопасность, быстродействие, а также поддержка *взаимодействия в режиме реального времени*. Возникновение этих ограничений обусловливается спецификой задачи, для решения которой применяется сеть.

Задача взаимодействия по сети в режиме реального времени – это обеспечение предопределенной верхней границы на время доставки сетевого пакета от одного узла сети до другого, включая все промежуточные узлы. Она имеет ряд актуальных примеров:

- VoIP (IP телефония) и интерактивное видео.
- Управление устройствами с обратной связью, например, роботы, где ключевую роль играют время доставки команд и время отклика.

При решении этой задачи в сетях с пакетной коммутацией возникают две подзадачи. Во-первых, время прохождения сетевого пакета между узлами сети может варьироваться в зависимости от длины пути (длина которого непостоянна) и задержек во внутренних очередях маршрутизаторов. Для решения этой задачи в современных сетях применяется резервирование сетевых ресурсов, классификация трафика и соответствующее планирование обработки пакетов по классам. Во-вторых, время обработки на конечных узлах пути, внутри сетевой подсистемы ОС, также должно быть предсказуемым.

Хорошим решением подобных задач могло бы стать наличие сетевого стека,

который можно было бы настроить под конкретную задачу, использовать на программных маршрутизаторах и на обычных хостах, и который бы обеспечивал обработку в режиме реального времени для высокоприоритетных видов трафика и QoS.

Одной из ОС, предоставляющих модульность и высокую конфигурируемость, является ОСРВ Embox, разрабатываемая уже более трех лет на кафедре системного программирования СПбГУ. В силу своей высокой конфигурируемости ОС Embox может работать как на обычных десктопах с архитектурой x86, так и на устройствах с сильно ограниченным объемом памяти, таких как Lego NXT. Высокая конфигурируемость всей системы в целом и послужила фундаментом для решения попробовать реализовать такой стек и в ОС Embox.

Необходимо сказать, что в ОС Embox первая версия сетевого стека появилась около трех лет назад, а первая версия модульного стека появилась немногим больше года назад. Так что в этом году у меня была работающая сетевая подсистема, и я занимался тестированием и анализом существующего решения.

Постановка задачи

Когда речь идет о системе реального времени, требуется некая гарантия того, что данная система действительно таковым свойством обладает. Для демонстрации этого факта касательно сетевой подсистемы ОСРВ Embox было предложено:

- Протестировать ОСРВ Embox в роли хоста и маршрутизатора
- Проверить на практике удовлетворение существующим стеком требованиям реального времени

Обзор существующих решений и подходов

Netgraph

Критерию гибкой конфигурируемости отвечает сетевая подсистема Berkley Netgraph, которая успешно применяется в операционных системах семейства *BSD. Netgraph – это сетевая подсистема в ядре, следующая принципу UNIX достижения мощности посредством комбинации простых инструментов, каждый их которых предназначен для выполнения одной, вполне определенной задачи. Основная идея проста: есть узлы (nodes) (инструменты) и ребра (edges), которые соединяют пару узлов (отсюда и "граф" в "netgraph"). Пакеты данных идут в двух направлениях вдоль ребер от узла к узлу. Когда узел получает пакет данных, он обрабатывает его и затем, как правило, отправляет его другому узлу. Обработка может быть простейшим добавлением/удалением заголовков или может быть более сложной, задействующей другие компоненты системы. В последних версиях Netgraph сетевые пакеты могут снабжаться специальной меткой приоритета, обозначающей, что данный пакет пройдет по графу быстрее менее приоритетных. Кроме того, Netgraph предоставляет обширную систему фильтрации пакетов, из-за чего FreeBSD так популярна как серверная ОС.

Из Netgraph были позаимствованы архитектурные решения, но портирование Netgraph в ОС Embox было бы очень трудоемким из-за сложности портирования любого весомого компонента из ядра в ядро. Кроме того, размеры модулей в BSD могут в десятки раз превышать размеры модулей в Embox, что сделало бы невозможным портирование Embox на устройства с ограниченными ресурсами. И наконец, третьим недостатком является отсутствие достоверной информации о том, обеспечивает ли Netgraph работу в режиме реального времени.

QNX

У Netgraph также имеется более маленький real-time аналог – сетевая подсистема популярной ОСРВ QNX. Помимо гибкого соединения узлов, инкапсулирующих в себе

реализации сетевых протоколов, в граф, QNX предоставляет также гибкую настройку стека на самом низком уровне из пространства пользователя (например, с помощью опции “-b brk_buf_numk” утилиты Net можно задавать размер RAM, отводимый для буферов хранения входящих пакетов). Однако исходный код этой сетевой подсистемы закрыт в силу проприетарности самой ОС.

RTNet

Третьим рассмотренным вариантом стека является RTNet (Hard Real-Time Networking for Real-Time Linux). Это сетевой стек реального времени для Xenomai и RTAI (real-time расширения Linux). Предоставляет детерминированные реализации UDP/IP, TCP/IP. Может использоваться в сетях Ethernet, так как реализует протокол RTmac для борьбы с коллизиями. Но он сочетает в себе два вышеуказанных недостатка BSD и QNX – большой размер и неподходящая лицензия.

Embox

В основе сетевой подсистемы ОСРВ Embox лежит определенный механизм – граф коммуникационных каналов.

В графе функциональные модули сетевого стека являются вершинами, коммуникационные каналы между модулями – ребрами. Вершины могут иметь несколько входящих и исходящие дуг. Также вводится понятие порта – точки крепления дуги к вершине (аналог hooks в Netgraph). Для каждой пары дуга-вершина существует порт, что позволяет вершине отличать конкретную дугу среди других дуг такого же типа (входящих или исходящих).

Также граф обладает входной вершиной, в которую попадают все пакеты, приходящие на обработку ОС Embox, и выходной, при достижении которой пакет считается полностью обработаным.

Такая организация предоставляет возможность модульной настройки сетевого стека. Заданием ребер графа осуществляется конфигурация обработки пакетов, коммутация, задание приоритетов и распределение пропускной способности и нагруженности каналов

связи.

Обработка каждого входящего пакета заключается в его прохождении от входной вершины, затем, согласно дугам, по промежуточным вершинам до выходной вершины. В процессе обработки узлы графа могут изменять пакет, проходящий сквозь них, а также выбирать дугу, по которой пакет продолжит свое движение.

Вершины, которые проходит пакет во время обработки, называются путем, или маршрутом пакета.

Возможно составить конфигурацию сети, при которой для определенных пакетов (например, приходящих с одного сетевого интерфейса) существует ровно один возможный путь. Тогда можно говорить о виртуальных каналах – предсказуемом пути обработки определенных пакетов.

Дуги обладают дополнительным атрибутом – приоритетом. Приоритет позволяет обрабатывать определенные пакеты раньше, чем другие, несмотря на порядок их получения. Таким образом, возможно сконфигурировать приоритетные каналы. Если узлы в этом канале обладают известным временем работы, то можно рассчитать гарантированное время обработки определенных пакетов.

Указанием приоритетов дуг каждому маршруту может быть назначен собственный приоритет. С помощью различных приоритетов пакетов обеспечивается максимально быстрая доставка пакетов, резервирование полосы пропускания сетевого устройства, а также защита от временных перегрузок в сети или на каком-либо сетевом интерфейсе.

Каждый модуль сетевой подсистемы выполняет определённую функцию и представлен в виде отдельного модуля сборки в системе. Таким образом, модули сетевого стека могут быть дополнительно настроены, им возможно задать требуемые ограничения и характеристики; некоторые из них могут не включаться в сборку.

Ввиду того, что сетевой стек Embox уже имеет высокую модульную гранулярность и, как следствие, высокую конфигурируемость, было решено тестировать и расширять существующий стек Embox, а не портировать стеки из других ядер, рассмотренные выше. Эти соображения и привели к постановке задачи.

Реализация

Когда приходит сетевой пакет, на сетевой карте происходит прерывание, и вызывается обработчик соответствующего драйвера сетевой карты, после чего пакет ставится в общую глобальную очередь. После выхода из первичного обработчика прерывания вызывается специальная функция, которая достает все пакеты из очереди и посылает их на вход графу. Первым выполняется классификатор – специальный узел, который сопоставляет заголовки пакета с набором заранее заданных правил и назначает новый приоритет пакету на основе приоритета, записанного в правиле. На каждый приоритет отводится свой поток в ядре ОС. После изменения приоритета пакет помещается в очередь потока с соответствующим приоритетом.

Тестовый полигон

В качестве тестового полигона был выбран симулятор QEMU для платформы x86. Несколько таких виртуальных машин, объединенных в одну сеть, позволяют быстро задавать нужную конфигурацию и быстро проводить новые замеры, внося исправления в исходный код. Измерения производились с отключением аппаратной виртуализации. Это было необходимо для того, чтобы QEMU полностью эмулировал ЦПУ и периферию, что обеспечивает достоверность измерений. Поэтому на реальном железе соответствующее поведение будет таким же, а цифры будут на несколько порядков меньше. Более детальное описание QEMU и сравнение с другими эмуляторами можно найти в статьях из используемой литературы.

Я измерял следующие показатели:

- Общее время прохождения пакета от точки входа до точки выхода из графа (синим цветом на графиках)
- Время, которое пакет реально обрабатывается внутри узлов (зеленый график)
- Время, которое занимает обработка прерываний, включая первичные и вторичные обработчики, то есть то время, что работающий поток простаивает (красный график)

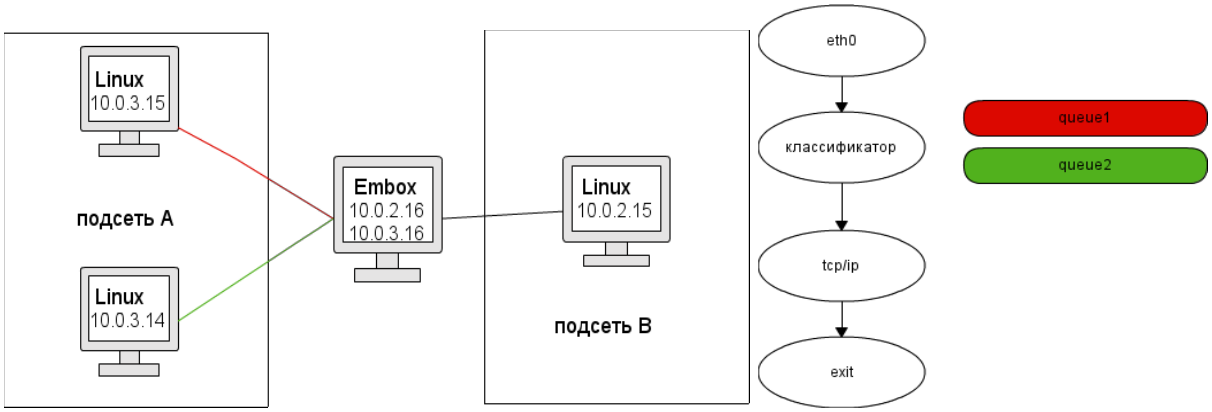
Все графики были посторены в GNU Octave – свободной системе для математических вычислений. По вертикальной оси отсчитывается время в микросекундах. По горизонтальной оси – порядковый номер высокоприоритетного пакета от 1 до 100.

Сценарий: маршрутизатор

Для тестирования стека Embox в качестве маршрутизатора использовалась следующая конфигурация.

Четыре виртуальные машины. На трех из них запущен образ linux-0.2.img, маленький образ ОС Linux, созданный специально для QEMU (его можно скачать по ссылке http://wiki.qemu.org/Testing). Две из этих машин объединяются в одну подсеть A, третья машина находится в подсети B. Четвертая машина с Embox имеет два сетевых интерфейса eth0 и eth1, и выполняет роль программного маршрутизатора между подсетями A и B. Обе машины a1 и a2 из подсети A с заданной периодичностью посылают пакеты разных приоритетов машине из подсети B. Я произвел замеры времени, необходимого ОСРВ Embox для маршрутизации. Проводилось несколько измерений для разных периодичностей отправки (в пределах 500-50000 микросекунд) для 100 пакетов высокого и 100 пакетов низкого приоритета.

Ниже приведена картинка топологии сети (слева), на которой красным и зеленым отмечены низкоприоритеное и высокоприоритеное соединения. На картинке справа изображена соответствующая структура графа в Embox. Любой входящий пакет проходит путь от вершины “eth0” до “exit”. Классификатор – это узел выполняющий классификацию пакетов по приоритетам на основе информации в заголовке, например, по IP адресу. Каждому соединению (красному и зеленому) в Embox соответствует своя очередь “queue1” и “queue2”, обозначенные красным и зеленым цветами соответственно.

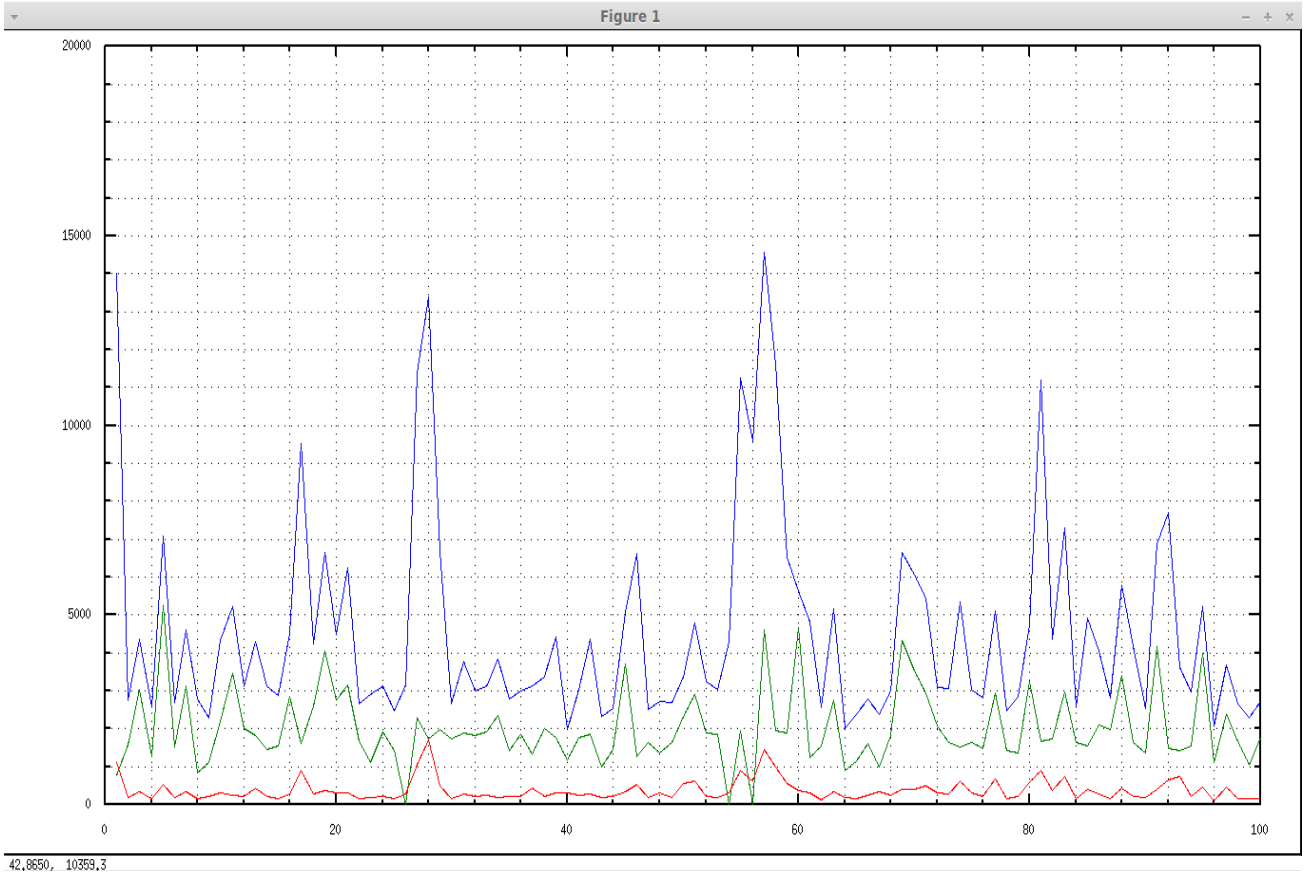


Фрагмент кода, исполняемого на хостах с Linux, осуществляющий отправку:

```
for (i = 0; i < 100; ++i) {
      char buf[0x10] = "Hello Embox!";
      if (sendto(sockfd, buf, sizeof buf, 0, (struct sockaddr *)&dst,
                  sizeof dst) < 0) {
            printf("ERROR sendto\n");
            break;
      } else {
            printf("(%d) sent %d bytes\n", i, sizeof buf);
      }
      usleep(1000);
}
```

Графики строятся для случая, когда пакеты отправлялись с интервалом 1 мс.

Результаты измерений

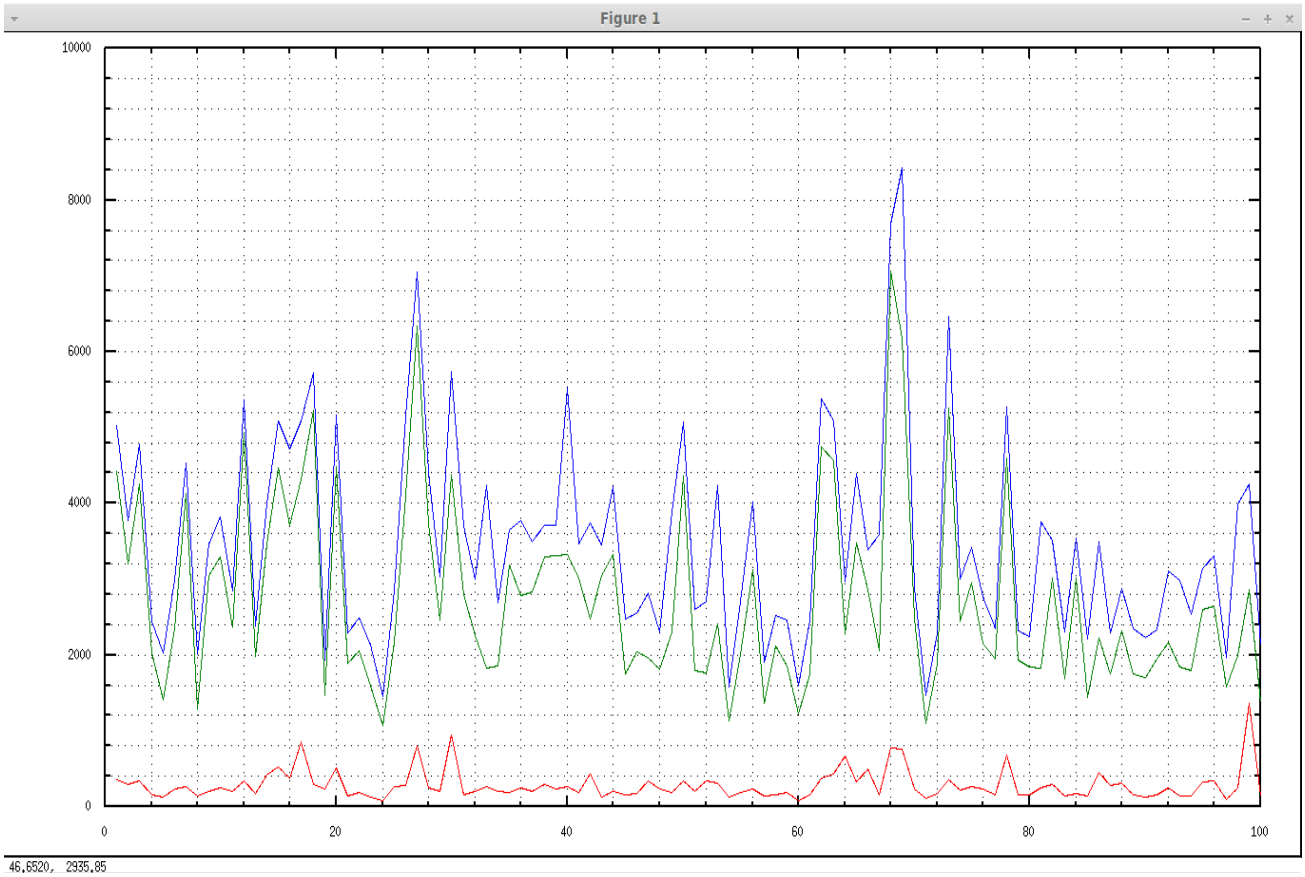


Как видно, обработка прерываний (красная ломаная) занимает малую часть от суммарного времени работы – не больше 2000 микросекунд за время обработки одного пакета. Также было установлено, что непостоянство этой ломаной обусловливается вложенностью прерываний, так например, на 28-м пакете произошло повторное прерывание от сетевой карты.

Непостоянство синей ломаной обусловлено ожиданием в очередях, так как алгоритм планирования для пакетов внутри очереди одного приоритета обрабатывает пакеты равномерно, то есть сначала все пакеты проходят первый обработчик, затем они все проходят следующий и т.д. Эта очередь имеет непостоянный размер, так как некоторые пакеты получают низкий приоритет и переставляются в другую очередь, а тем временем приходят новые пакеты и снова увеличивают приоритетную очередь.

В подтверждение моим словам ниже приведен график, на котором зеленым цветом

обозначено (время обработки + время ожидания в очередях):



На этом графике мы видим, что действительное время обработки пакетов наивысшего приоритета почти совпадает с общим временем в моменты всплесков, за исключением времени переключения потоков и некоторых вызовов функции memcpy, даже в условиях загруженности сети низкоприоритетным трафиком. Время переключения потоков по некоторому событию также было измерено и варьируется в пределах 100-300 микросекунд. Представленные графики зависят от длины очереди обрабатываемых пакетов, которая может быть достаточно большой, если пакеты отправляются очень часто.

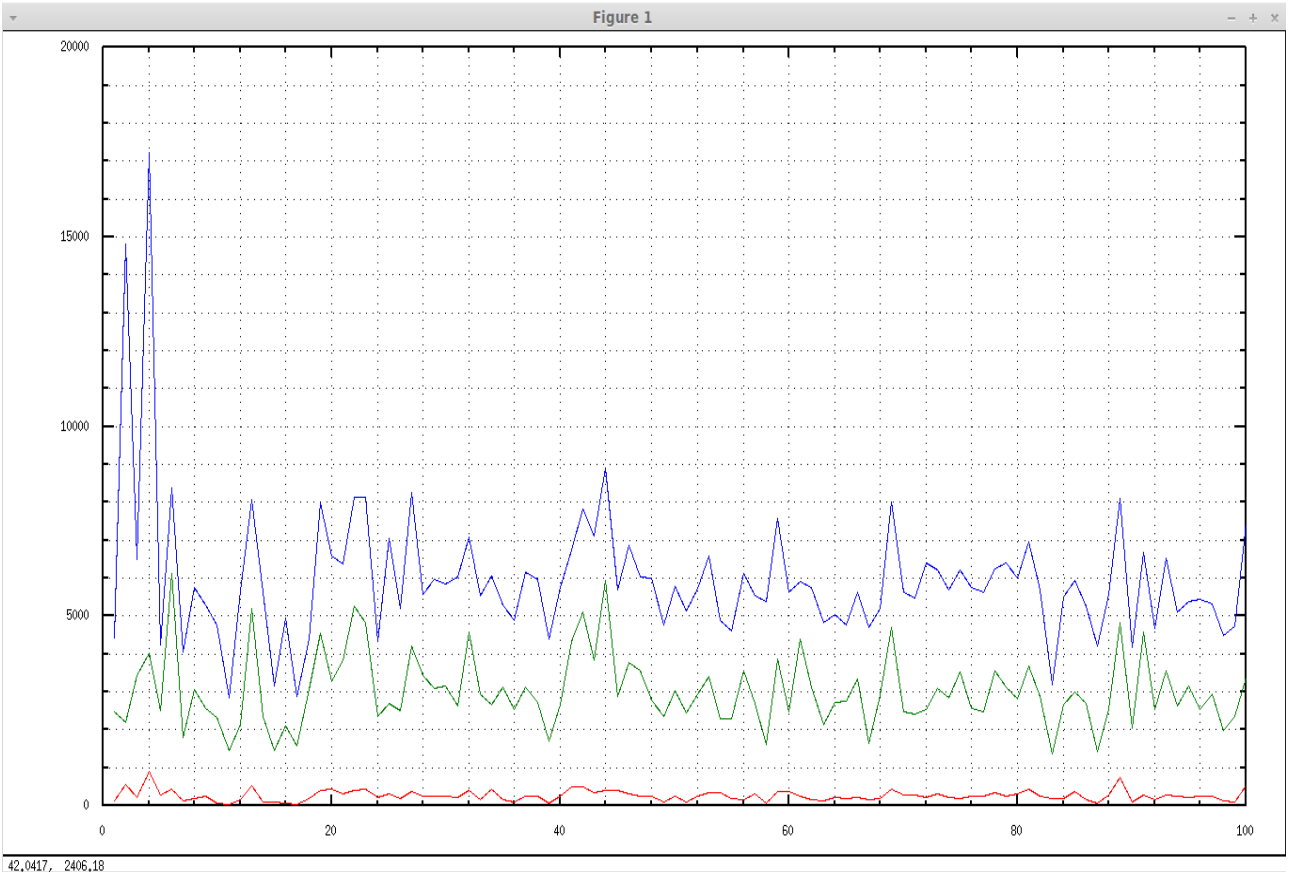
Сценарий: хост-получатель

Также были проведены тесты для Embox в роли хоста-получателя. Для этого использовалась конфигурация сети, аналогичная конфигурации для маршрутизации описанной выше. В этом случае отсутствует подсеть B, есть только подсеть A, по-прежнему состоящая из двух машин с Linux, и в этой же подсети находится Embox с

одной сетевой картой. Машина с Embox обрабатывает пакеты, приходящие от двух соседних хостов, по той же схеме, что и в случае маршрутизации.

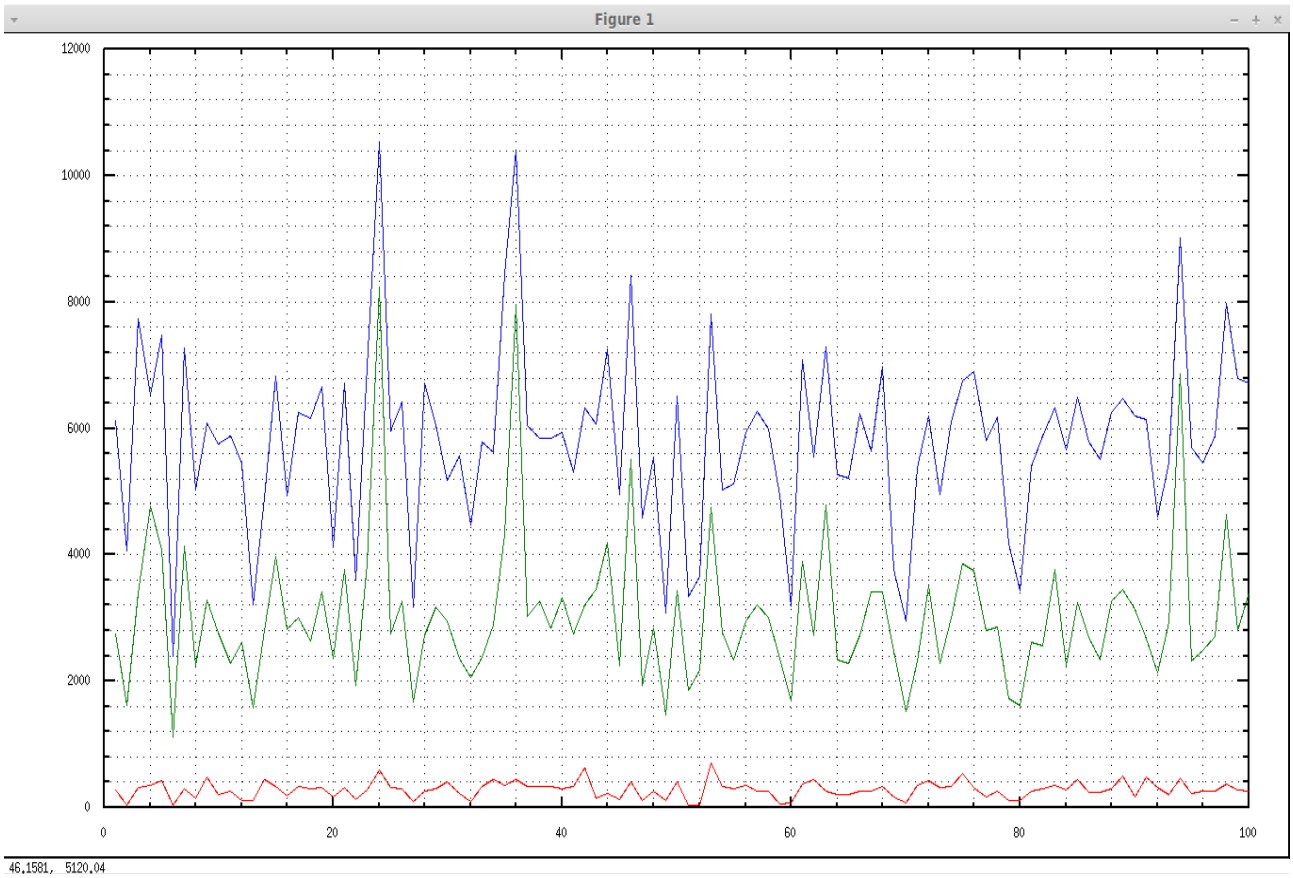
Результаты измерений

График для периодичности отправки пакетов 1 мс представлен ниже:



Синяя ломаная, как и ранее, обозначает суммарное время обработки пакета. Зеленая – время, которое пакет реально обрабатывался внутри узлов. А красная – это время проведенное внутри обработчиков прерываний.

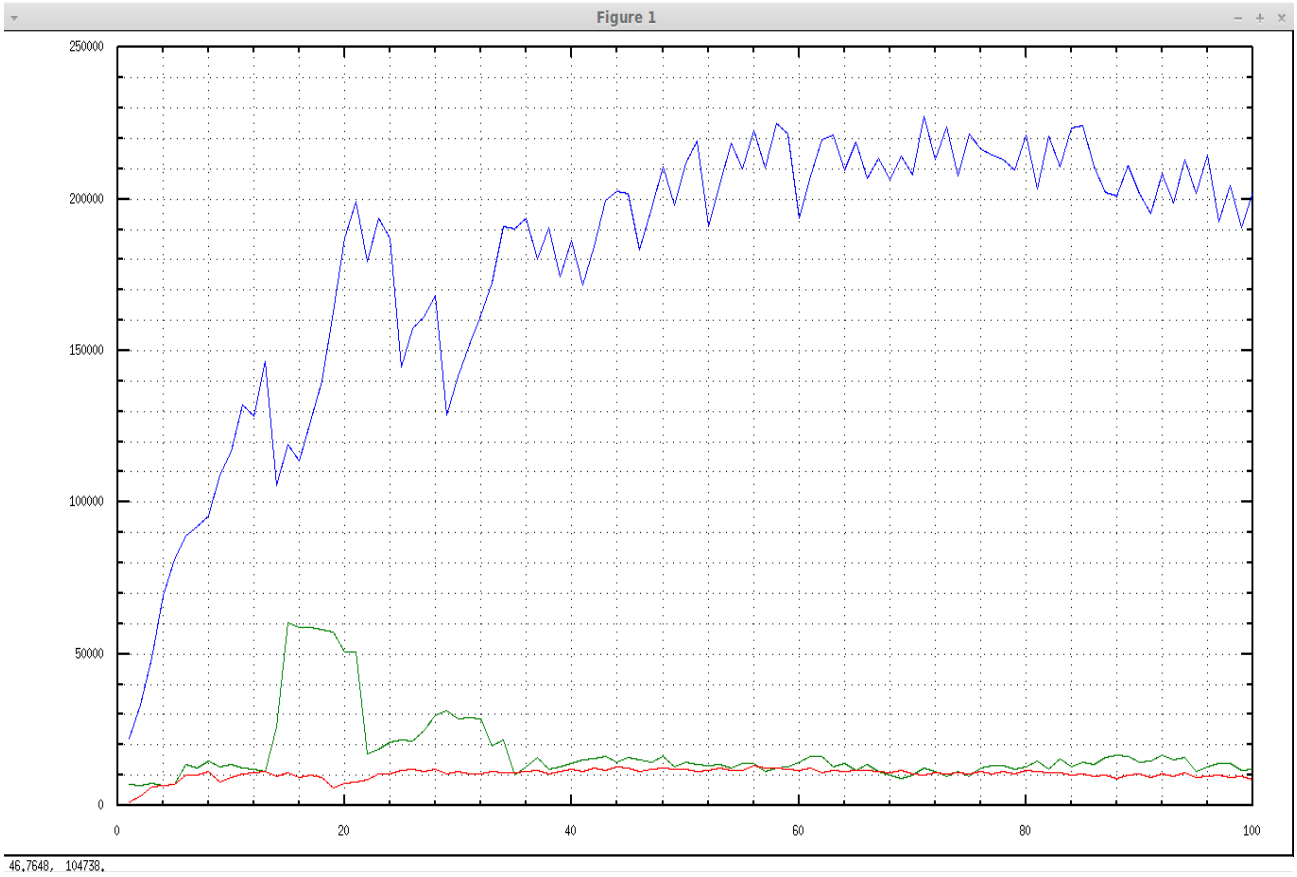
Ниже приведен еще один график для интервала отправки 50 мс:



На графике видно, что время обработки пакета почти не изменилось. Это вызвало подозрения и выявило то, что текущий вариант TCP в Embox не справляется с нагрузкой, когда частота отправки составляет 1 мс. В подтверждение этому были также выявлены потери пакетов. Однако, верхняя граница времени обработки пакета TCP не изменяется и остается равной 20 мс.

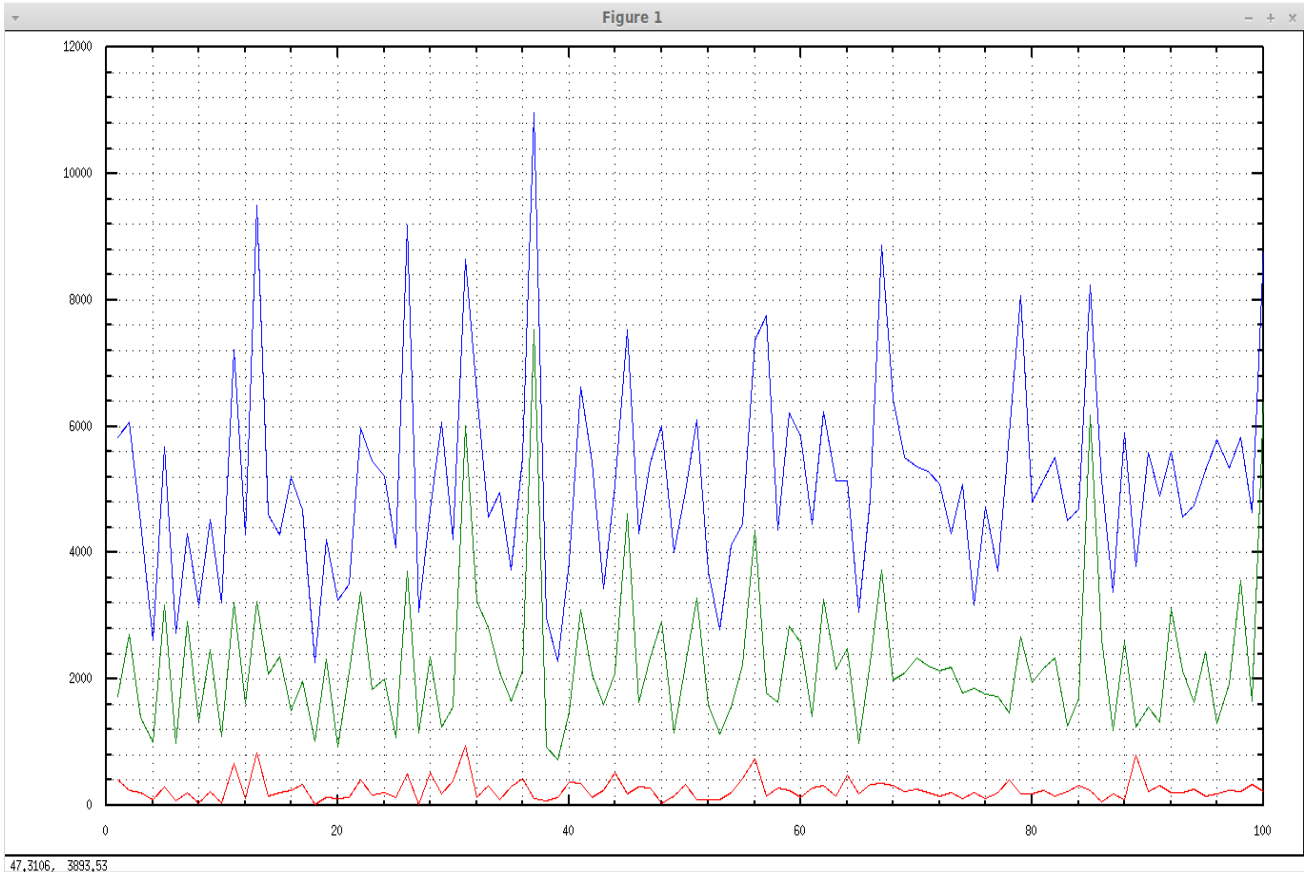
В дополнение к этому были проведены те же тесты, но поверх протокола UDP. Ниже приведены два графика:

Первый график для случая отправления пакетов с частотой 1 мс:



На графике видно, что сначала время небольшое и постепенно возрастает. Это вызвано тем, что изначально очередь пакетов пуста, и входящим пакетам не приходится ожидать своей очереди, поэтому они обрабатываются быстро. Заметим, что время обработки составялет порядка 20 мс, а это значит, что при частоте отправления пакетов 1 мс, очередь будет неизбежно наполняться. Поэтому каждый следующий пакет стоит в ожидании большую часть времени обработки, при этом время стабилизируется.

При частоте отправления 50 мс отношение общего времени (синия ломаная) к реальному времени обработки (зеленая ломаная) в разы меньше, что подтверждает тот очевидный и ожидаемый результат, что при большем периоде отправления пакеты в очереди накапливаются менее интенсивно. Ниже представлен график:



Подводя итог, верхняя граница времени обработки высокоприоритетного сетевого пакета в сетевом стеке ОСРВ Embox не зависит от остального трафика и является функцией, зависящей от частоты поступления высокоприоритетных пакетов. Таким образом, если ограничить период получения высокоприоритетных пакетов сверху некоторой константой, скажем, 1 мс, то получится верхняя граница на время обработки пакета вне зависимости от нагрузки, создаваемой остальным трафиком. Так для маршрутизации TCP она составила 30 мс для разных частот отправки. Для UDP – 25 мс для частоты 1 мс и 15 мс для частоты получения 50 мс. Для хоста-получателя по протоколу TCP – 20 мс.

Реализованные технолигии

Для тестирования производительности был использован профайлер в ядре Embox, реализованный в прошлом году в качестве курсовой работы одним из участников нашей команды. Этот профайлер позволяет обрамлять участки кода trace-блоками, которые включают два tracepoint'а – точка входа и точка выхода. Для решения поставленной задачи его пришлось модифицировать. Добавлена возможность задавать условия, которые проверяются на входе trace-блока, и если условие выполнено, то trace-блок выполняется. Это позволило измерять время обработки прерываний для конкретно выбранного потока, а не суммарное. Для этого понадобилось добавить небольшую опцию - получение trace-block'а и tracepointa'а по имени вне файла, где он определен.

Также был реализован простой сэмплирующий профайлер, который умеет собирать информацию о количестве вызовов функции. В любой момент времени в ОС можно получить текущий sample с помощью вызова функции **sampling_copy_sample**, которая записывает sample в буфер. Получив такие samples в разные моменты времени, можно посчитать их разницу воспользовавшись функцией **sampling_diff_samples**. Этот механизм был реализован мной для того, чтобы выяснить несовпадение синей и зеленой ломаных в приведенных графиках.

Для замеров был реализован тестовый фреймворк, который предоставляет возможность логирования результатов в файл внутри файловой системы Embox, функции **pnet_timer_hnd_enter**, **pnet_timer_hnd_leave,** и **pnet_timer_hnd_stopped,** которые являются аналогами trace-блоков, но могут размещаться в различных файлах и модулях, что позволило найти суммарное время выполнения всех обработчиков сети (зеленая ломаная). Функция **pnet_timer_hnd_stopped** вызывается в конце обработки пакета, что приводит к записи накопленной информации в файл. Сама информация хранится внутри служебной структуры, которая заполняется по мере прохождения пакета по графу. Ниже приведен фрагмент кода, демонстрирующий пример использования этих функций:

```
        …
if (NULL != node->proto) {
        if (NULL != hnd) {
```

```
                    /* pnet_timer_hnd_enter вызывается в другом файле  */
                    res = hnd(pack);
                    pnet_timer_hnd_leave(pack);
              }
              ...
       }
              …
case NET_HND_STOP:
       pnet_timer_hnd_stopped(pack);
```

Результаты

В ходе работы над курсовой я получил следующие результаты:

- Реализован фреймворк для тестирования сети
- Реализован примитивный сэмплирующий профайлер и сделаны вспомогательные доработки в существующем профайлере Embox
- Произведены замеры производительности для случаев разных приоритетов входящих пакетов, для разных частот отправки и в двух конфигурациях Embox – в роли маршрутизатора и в роли конечного получателя. Опытным путем определена верхняя граница времени обработки.

Дальнейшие планы

В дальнейшем хочется реализовать в Embox гораздо более содержательный профайлер типа LTT (Linux Trace Toolkit), который позволяет собирать трассировки в ядре Linux из основных подсистем ОС – подсистема памяти, сетевая и файловая подсистемы и т.д. Если сделать этот механизм так, чтобы он выдавал результаты в том же формате, что и LTT, то можно использовать существующие инструменты, чтобы визуализировать результаты, так же как это делается в Linux. Это позволит:

- произвести сравнение Embox и real-time Linux
- протестировать все подсистемы и обнаружить узкие места в их производительности

Используемая литература

1. Netgraph in 5 and beyond - http://people.freebsd.org/~julian/BAFUG/talks/Netgraph/Netgraph.pdf
2. QNX networking - http://www.qnx.com/solutions/industries/netcom/#technology
3. QEMU - http://qemu.weilnetz.de/qemu-tech.html#intro_005fx86_005femulation
4. Linux Trace Toolkit - http://www.opersys.com/ltt/
5. RFC