Qualidade de Serviço em Redes utilizando Linux em Sistemas Embarcados

Ricardo da Silva Souza, Andrelise Rafael Gonçalves, Caio César Hollerbach Gomes, Bruno Reis Côrrea

***Abstract*— This document contains information for configuring an embedded system with Linux operating system running as an access point with quality of service by dividing the traffic on a TCP / IP network in different classes both in rate values as in priority, and also presents the tests for proving the efficiency of this resource.**

***Index Terms*—Embedded System, Network, Quality of Service.**

***Resumo*—Este documento contém informações para a configuração de um sistema embarcado com sistema operacional Linux operando como ponto de acesso com qualidade de serviço, dividindo o tráfego em uma rede TCP/IP em classes diferenciadas tanto em valores de taxa quanto em prioridade, e, também, apresenta os testes para a comprovação da eficiência do deste recurso.**

***Palavras chave*—Redes, Sistemas Embarcados, Qualidade de Serviço.**

I. Introdução

As redes TCP/IP (*Transmission Control Protocol/Internet Protocol*) não foram criadas para atender a demanda das aplicações atuais, não fornecendo nativamente recursos para tratar problemas de QoS (*Quality of Service*), atualmente a Internet oferece apenas conectividade básica sem garantia ou controle de seus serviços, porém este modelo atende as exigências da maioria dos serviços convencionais oferecidos na rede como transferência de arquivos, correio eletrônico e navegação WEB uma vez que possui controle de sequenciamento e perda de pacotes.

Com o constante crescimento da Internet e a integração de aplicações multimídia nas redes, se faz necessário o tratamento diferenciado dos serviços para os diversos tipos de tráfego existentes, atualmente serviços de vídeo e voz que possuem um tráfego sensível a atraso disputam banda com serviços de transferência de arquivo igualmente sem qualquer diferenciação.

Para controlar o tráfego da rede deve-se requisitar este serviço ao ISP (*Internet Service Provider*) da rede ou configurar um dispositivo com estes recursos.[1]-[2]

Uma vez que os roteadores de baixo custo desenvolvidos para redes SOHO (*Small Office and Home Offices*), modelo de rede apresentado na Figura 1, oferecem apenas recursos básicos como NAT (*Network Address Translation*) e DHCP (*Dynamic Host Configurarion Protocol*), para realizar o controle e diferenciação do tráfego sem a necessidade de contratar qualidade de serviço no ISP da rede ou requisitar um novo aparelho que possua este recurso pode-se utilizar de recursos do SO (Sistema Operacional) Linux.[3]

Um computador devidamente configurado pode oferecer este serviço, assim como, também, qualquer outro dispositivo com SO Linux, porém a utilização de um computador para tal serviço acaba por desperdiçar recursos de *hardware*, quando pode-se utilizar dispositivos dedicados para atender este recurso.

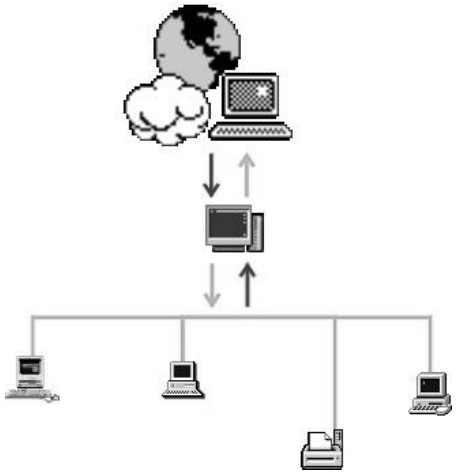

Fig. 1. Modelo de rede SOHO.

Este trabalho contempla o desenvolvimento de um sistema embarcado para atender este cenário, este consiste em um ponto de acesso para uma rede SOHO utilizando o SO Linux para gerenciar e controlar o tráfego na rede.

Composto de uma interface ethernet para a conexão com a rede local ou ISP e outra *wireless* operando em modo AP (*Access Point*) provendo qualidade de serviço na rede para todos os hosts conectados.

II. Kit de Desenvolvimento

O hardware para construção deste sistema é um kit de desenvolvimento mini2440 da fabricante Friendly ARM, este equipamento possui uma CPU (*Central Processing Unit*) S3C2440 da Samsung baseado no core ARM920T RISC (*Reduced Instruction Set Computing*) com 400Mhz de clock, 64Mb de SDRAM (*Synchronous Dynamic Random Access Memory*), 256Mb de NAND flash e 2Mb de NOR flash.

A memória flash é um chip de armazenamento não volátil que pode ser apagado e reprogramado eletricamente, o tipo

NAND é usado em USB (*Universal Serial Bus*) flash drives, cartões SD (*Secure Digital*) e SSDs (*Solid State Drive*), este tipo de memoria tem maior capacidade de armazenamento e permite apenas acesso paginado, já as memórias do tipo NOR permite acesso randômico para leitura como nas memórias RAM (*Random Access Memory*), sendo assim, uma memória mais rápida em comparação a NAND, mas, em contrapartida possuem capacidade de armazenamento menor. Uma particularidade desta placa é a opção de iniciar pela memória NAND ou NOR através de uma chave de seleção.[4]

O equipamento oferece entrada e saída de audio, RTC (*Real Time Clock*), Buzzer PWM (*Pulse-Width Modulation*), interfaces para conexão USB, ethernet, serial RS232, JTAG (*Joint Test Action Group*), slot para cartão SD, interface para display LCD (*Liquid Crystal Display*) e câmera em uma placa de 10x10 cm como mostrado na Figura 2:

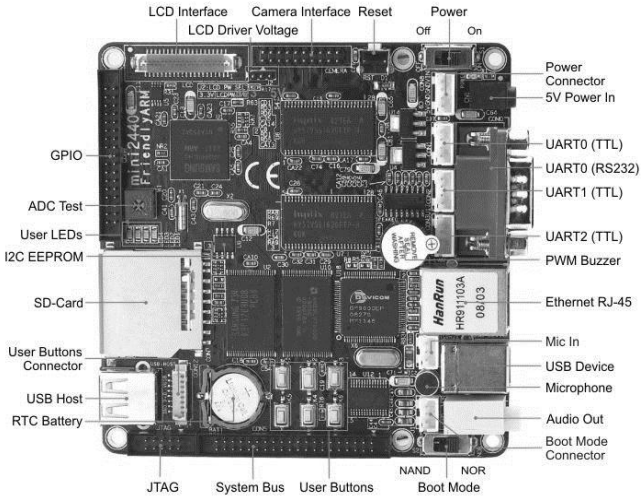

Fig. 2. Placa de desenvolvimento e suas interfaces I/O (Input/Output).

A figura 3 mostra o kit de desenvolvimento completo. Este inclui um display LCD *touch screen* de 3,5" polegadas, cabo serial RS232, JTAG, USB, fonte de alimentação DC (*Direct Current*) 12V e vem configurado de fábrica com o SO Linux, GUI (Graphical User Interface) Qtopia e bootloader Supervivi.[5]-[7]

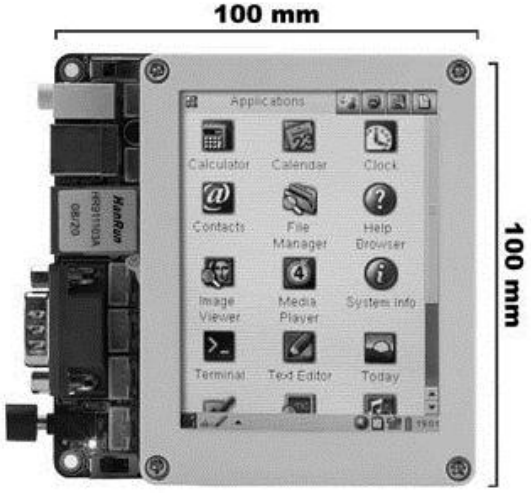

Fig. 3. Kit de desenvolvimento Friendly Arm mini2440 completo.

III. Ambiente de Desenvolvimento

O desenvolvimento é realizado em um notebook com a distribuição do SO Linux Ubuntu na versão 11.10 Oneiric Ocelot. O acesso ao console do equipamento é feito através da interface serial do equipamento usando um cabo RS232 fornecido pelo kit e um adaptador serial-usb para a comunicação com o notebook, e o aplicativo minicom devidamente configurado para comunicar com a interface USB utilizada e velocidade 115200bps.

A toolchain com os cross-compiladores e as libs necessárias para compilação, assim como, também, o kernel e a aplicação usb-push são fornecidas na página do fabricante[8].

A comprovação de resultados utiliza a aplicação iperf, esta é capaz de gerar tráfego na rede passando como parâmetro protocolos, taxas e portas, assim como, também, pode fazer a leitura de uma determinada porta na rede.

IV. Bootloader

O bootloader é a aplicação responsável pela inicialização do hardware e por carregar aplicações na RAM, como o kernel do SO. Também oferece a capacidade de gravar aplicações na flash através de comunicação USB ou ethernet via TFTP (*Trivial File Transfer Protocol*) por exemplo.

O DAS U-Boot, é um projeto *open source* amplamente conhecido em sistemas embarcados, possui recursos avançados e grande comunidade de desenvolvimento, além de ser um bootloader leve e com suporte a diversas plataformas como ARM, PowerPC, x86.[9]

V. Sistema Operacional

Sistema Operacional é um conjunto de aplicações e scripts para gerenciar os recursos do sistema e realizar a comunicação entre as camadas de software e hardware. Ele realiza a gerência de processos, memória, sistema de arquivos e dos dados.

Um sistema operacional multitarefa é capaz de gerenciar os recursos da CPU de forma que vários processos sejam executados por partes, a execução destes é tão rápida que dá ao usuário a impressão de que esta sendo realizada simultaneamente. A comunicação entre estes processos é chamada de IPC (*Inter-Process Communication*).

A memória é gerenciada pelo SO de forma que cada processo seja endereçado sem ocorrer relocação de memória, protegendo o espaço de cada para que não utilize um endereço de memória já utilizado por outro processo, ele também possibilita o swapping, fazendo com que um processo utilize mais memória do que a existente no sistema, utilizando uma área da memória secundária ou memória não volátil para endereçar este processo.

O acesso ao sistema de arquivos também é gerenciado pelo SO, este é o responsável por armazenar e recuperar informações.

O Linux possui um kernel e um rootfs, estes dois trabalham juntamente para gerenciar todo o sistema, o kernel é o núcleo

do SO e o responsável pela comunicação do hardware com software.

Na Figura 4 pode-se observar como este comunicação é realizada, na camada mais alta, temos as aplicações e os drivers, que são responsáveis por comunicar ao kernel como controlar um determinado hardware no sistema e criar a interface entre o espaço do usuário e o espaço do kernel, na camada intermediária encontra-se o rootfs e os protocolos de controle do sistema circundados pela camada de alto e baixo nível do kernel. A camada de alto nível no Linux gerencia as aplicações e seu código normalmente é escrito na linguagem de programação C, enquanto a camada de baixo nível gerencia a parte física do sistema, controla as interfaces e periféricos do equipamento e seu código é escrito em Assembly, estas duas camadas se comunicam entre si, através dos protocolos do sistema.

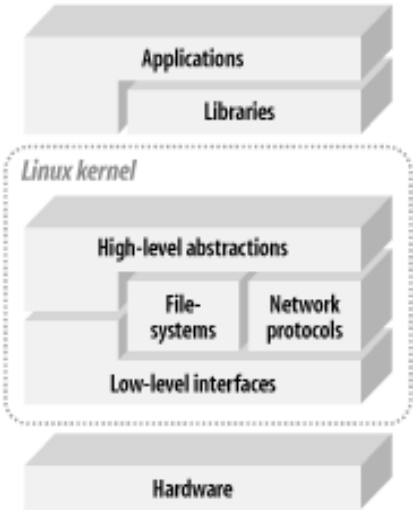

Fig. 4. Funcionamento do sistema operacional

Quando iniciado o sistema, o kernel é descompressado e carregado na RAM, neste momento ele passa a enviar mensagens ao terminal com o status de sua execução. Quando carregado com sucesso acessa o rootfs e executa o init, que por sua vez executa scripts para a inicialização dos serviços.[10]

O SO utilizado no sistema é o Debian na versão Squeeze, um sistema operacional open-source, ou seja, seu código pode ser uitlizado ou alterado por qualquer pessoa livremente sob a licença GPL (*General Public License*), sendo também estável, confiável e portável, o que facilita sua utilização no desenvolvimento em sistemas embarcados.

VI. Construção do Sistema

O sistema deve apresentar uma estrutura como a Figura 5:

Fig. 5. Estrutura do sistema

O SO é armazenado em um SD Card de 2GB, neste são criadas três partições, uma partição de swap com 512MB para endereçamento de processos quando não houver RAM disponível, as outras duas partições irão armazenar a imagem do kernel e o rootfs, normalmente a partição que armazena o kernel utiliza o sistema de arquivos ext2 por ser mais estável em comparação aos outros, enquanto a partição do rootfs utiliza um sistema de arquivos atual e com mais recursos, o ext3. São separados 50MB para o kernel e o restante para o rootfs.

O bootloader por sua vez é armazenado na memória flash NAND.

A. *Kernel*

O kernel em desenvolvimento está na versão 2.6.32. Este é compilado utilizando as configurações padrões do nosso kit de desenvolvimento e adicionados os módulos necessários para carregar o dispositivo wifi conectado a interface USB do equipamento além dos módulos necessários para o tratamento de pacotes na rede. Compilado o kernel com as ferramentas de cross compilação da toolchain, gera-se uma imagem do mesmo para ser carregada pelo bootloader, esta imagem é inserida na segunda partição do SD Card.

B. *Rootfs*

O rootfs é construído com a aplicação debootstrap, esta aplicação gera uma estrutura de diretórios e arquivos utilizando os pacotes pré-compilados de uma distribuição Debian para a arquitetura em desenvolvimento. Construída a estrutura, esta é inserida na terceira partição do SD Card.

Devem ser gerados também os módulos do kernel passando como caminho o rootfs.

C. *Bootloader*

A transferência do bootloader é feita pela interface USB do equipamento e do notebook através de um cabo USB fornecido pelo kit e a aplicação usb-push.

Para gravação do bootloader na flash NAND, conecta-se o equipamento ao computador pelo cabo serial RS232 e USB, a chave no equipamento é alterada para que o mesmo possa inicializar pela memória NOR, enquanto no terminal do computador o equipamento é acessado pelo minicom. Com o sistema iniciado pela memória NOR o bootloader SuperVivi se inicia e seu menu é exibido como mostrado na Figura 5.

```
##### FriendlyARM BIOS 2.0 for 2440 #####
[x] format NAND FLASH for Linux
[v] Download vivi
[k] Download linux kernel
[y] Download root_yaffs image
[a] Absolute User Application
[n] Download Nboot for WinCE
[l] Download WinCE boot-logo
[w] Download WinCE NK.bin
[d] Download & Run
[z] Download zImage into RAM
[g] Boot linux from RAM
[f] Format the nand flash
[b] Boot the system
[s] Set the boot parameters
[u] Backup NAND Flash to HOST through USB(upload)
[r] Restore NAND Flash from HOST through USB
[q] Goto shell of vivi
[i] Version: 0945-2K
Enter your selection:
```

Fig. 6. Menu do bootloader SuperVivi

No SuperVivi o U-Boot é enviado pela USB através da aplicação usb-push, este é escrito em uma posição da memória RAM, esta posição é passada para o bootloader para que o mesmo possa executá-la, surgindo o shell do U-Boot após sua inicialização, a memória NAND é limpada para que ele seja gravado, o bootloader é gravado no primeiro endereço da memória para que ele possa ser executado ao ligar o equipamento com sua chave de seleção na opção NAND.

Devem-se ser alteradas as variáveis de inicialização, os parâmetros de inicialização devem dizer como e onde será executado o SO.

Estas variáveis também são responsáveis por dizer ao bootloader o sistema de arquivos utilizado por cada partição que é carregada.

VII. Configuração do Sistema e Qualidade de Serviço

Em um dispositivo de rede normal, como em um roteador, todo pacote é aberto para que seu cabeçalho possa ser lido, a fim de encaminhá-los corretamente ao seu destino, seguindo uma tabela de rotas do dispositivo, porém ao utilizar qualidade de serviço, esta concepção vai um pouco além, após a leitura deste cabeçalho cada pacote recebe um tratamento diferenciado de acordo com as regras estabelecidas no dispositivo como mostrado na Figura 7.

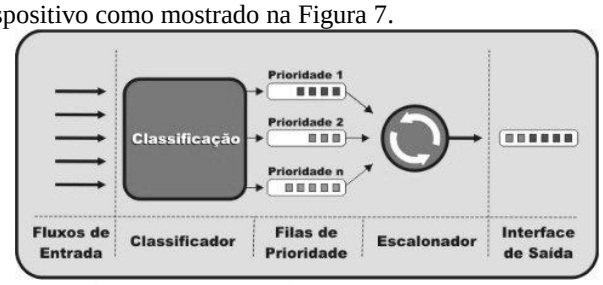

Fig. 7. Classificação de pacotes em filas.

Aplicar qualidade de serviço em uma rede interfere na concorrência de banda, dando prioridade a um tráfego ou pacote específico em detrimento de outros.

Outro recurso possível de adotar em qualidade de serviço, utilizado por ISPs, é a divisão da banda para vários hosts em uma rede. Da mesma forma que podemos interferir no comportamento do tráfego de pacotes individualmente, podemos também interferir em um fluxo de dados pré-definido.

Em QoS podemos adotar dois modelos básicos para controlar o tráfego na rede, estes são o policiamento e suavização.

Policiamento consiste em adequar um tráfego a uma determinada taxa descartando o excedente enquanto suavização trabalha com o armazenamento destes pacotes em buffers para evitar o descarte como mostrado na Figura 8.

Comparando os dois modelos possíveis, policiamento de tráfego é um recurso mais leve e barato, exigindo menos recursos de hardware para a sua implementação e atende as bem necessidades na maioria dos cenários de redes atuais, por outro lado a suavização otimiza a quantidade de dados transmitidos, pois reduz o descarte de pacotes, podendo ser usado como alternativa quando se deseja reduzir o descarte de pacotes.[11]

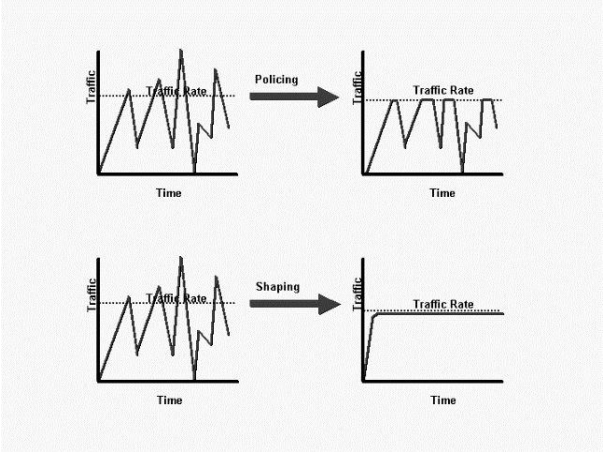

Fig. 8. Policiamento e suavização de tráfego.

A. *Wifi e Ponto de Acesso*

Para a interface Wifi operar em modo AP (Access Point) utiliza-se a aplicação hostapd, esta aplicação é capaz subir uma interface wireless compatível como ponto de acesso, recebendo como parâmetro em seu arquivo de configuração a chave de criptografia desejada, o SSID (*Service Set Identifier*), e, também o padrão de operação, o hostapd é configurado para iniciar na juntamente com o equipamento, para que a interface wireless seja configurada automaticamente a cada inicialização. A Figura 9 mostra a saída do iwconfig, onde são apresentadas as configurações da interface wifi operando como ponto de acesso.

```
Linux QoSGW 2.6.32.2 #1 Wed May 16 17:49:31 BRT 2012 armv4tl

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@QoSGW:~# iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

wlan0     IEEE 802.11bgn  Mode:Master  Frequency:2.462 GHz  Tx-Power=16 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Power Management:on

br0       no wireless extensions.

mon.wlan0  IEEE 802.11bgn  Mode:Monitor  Frequency:2.462 GHz  Tx-Power=16 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Power Management:on

root@QoSGW:~#
```

Fig. 9. Wifi como ponto de acesso

B. *Interfaces*

As principais ferramentas responsáveis por levantar as interfaces de rede, gerenciar as rotas, e, também, configura-las devidamente são o ifconfig e o route.

O arquivo interfaces mantém as configurações iniciais do sistema, utilizando estas duas ferramentas para executá-las.

Para o nosso sistema, uma bridge será iniciada e receberá um IP de um servidor DHCP (*Dynamic Host Configuration Protocol*), esta será a ponte entre as interfaces ethernet e wifi.

A interface ethernet é conectada a rede local enquanto a wifi opera como ponto de acesso para os hosts que serão providos de qualidade de serviço.

C. Configuração do tráfego

O gerenciamento do tráfego utilizada a aplicação iproute2, esta é um conjunto de ferramentas para controlar roteamento, controle de tráfego, e, também, gerencia redes IPv4 e IPv6.

A ferramenta TC (*Traffic Control*), disponibilizada pela iproute2, é a principal responsável na implementação de QoS no kernel Linux.[12]

Para fins de teste e comprovação, foram utilizadas as portas 5000, 5001 e 5002. Dividindo o tráfego destinado a estas, e, associando-as respectivamente a primeira, segunda e terceira classe.

Três classes são criadas com pesos diferenciados, em uma rede com taxa máxima de 10Mbps a primeira classe é a mais prioritária e é definida com um tráfego mínimo garantido de 5Mbps e máximo de 10Mbps, a segunda é definida entre 3Mbps e 10Mbps, e a terceira garante uma taxa de 2Mbps para o tráfego destinado à ela, sendo esta ultima a classe padrão da rede, todo o tráfego não direcionado a nenhuma outra classe será um tráfego desta ultima classe.

O tráfego dentro da mesma classe é dividido igualmente, para isso associa-se o algoritmo de gerenciamento de tráfego SFQ (*Stochastic Fairness Queuing*), este algoritmo garante uma divisão justa da taxa dentro da classe a qual ele é associado.

VIII. TESTE E COMPROVAÇÃO DOS RESULTADOS

Os resultados podem ser comprovados enviando tráfego do equipamento para o computador com a ferramenta iperf.

Três seções de terminais via SSH (*Secure Shell*) são criadas acessando o sistema como mostra a Figura 10.

```
root@QoSGW:~# iperf -c 192.168.20.125 -b 10M -p 5000 -i1 -t500
WARNING: option -b implies udp testing
------------------------------------------------------------
Client connecting to 192.168.20.125, UDP port 5000
Sending 1470 byte datagrams
UDP buffer size:   110 KByte (default)
------------------------------------------------------------
[  3] local 192.168.20.156 port 48516 connected with 192.168.20.125 port 5000
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec  1.19 MBytes  10.0 Mbits/sec
[  3]  1.0- 2.0 sec  1.19 MBytes  10.0 Mbits/sec
[  3]  2.0- 3.0 sec  1.03 MBytes  8.61 Mbits/sec
[  3]  3.0- 4.0 sec    843 KBytes  6.90 Mbits/sec
[  3]  4.0- 5.0 sec    672 KBytes  5.50 Mbits/sec
[  3]  5.0- 6.0 sec    576 KBytes  4.72 Mbits/sec
```

Fig. 10(a). Equipamento enviando tráfego na porta 5000

```
root@QoSGW:~# iperf -c 192.168.20.125 -b 10M -p 5001 -i1 -t500
WARNING: option -b implies udp testing
------------------------------------------------------------
Client connecting to 192.168.20.125, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:   110 KByte (default)
------------------------------------------------------------
[  3] local 192.168.20.156 port 43610 connected with 192.168.20.125 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec    432 KBytes  3.54 Mbits/sec
[  3]  1.0- 2.0 sec    369 KBytes  3.02 Mbits/sec
[  3]  2.0- 3.0 sec    356 KBytes  2.92 Mbits/sec
```

Fig. 10(b). Equipamento enviando tráfego na porta 5001

```
root@QoSGW:~# iperf -c 192.168.20.125 -b 10M -p 5002 -i1 -t500
WARNING: option -b implies udp testing
------------------------------------------------------------
Client connecting to 192.168.20.125, UDP port 5002
Sending 1470 byte datagrams
UDP buffer size:   110 KByte (default)
------------------------------------------------------------
[  3] local 192.168.20.156 port 38571 connected with 192.168.20.125 port 5002
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec    293 KBytes  2.40 Mbits/sec
[  3]  1.0- 2.0 sec    274 KBytes  2.25 Mbits/sec
```

Fig. 10(c). Equipamento enviando tráfego na porta 5002

Das três seções do são enviados pacotes de tamanho 10Mbps utilizando protocolo UDP (*User Datagram Protocol)* e com destino ao endereço IP do computador para as portas 5000, 5001 e 5002.

No computador os três terminais são criados para ler as mesmas portas e obter as informações do tráfego recebido, como mostra a Figura 11.

```
rssouza@rssouza-pc:~$ iperf -s -u -p 5000 -i1
------------------------------------------------------------
Server listening on UDP port 5000
Receiving 1470 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[  3] local 192.168.20.125 port 5000 connected with 192.168.20.156 port 48516
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0- 1.0 sec  1.16 MBytes  9.76 Mbits/sec   0.589 ms    0/  830 (0%)
[  3]  1.0- 2.0 sec  1.16 MBytes  9.73 Mbits/sec   0.544 ms    0/  827 (0%)
[  3]  2.0- 3.0 sec  1.01 MBytes  8.49 Mbits/sec   2.855 ms    0/  722 (0%)
[  3]  3.0- 4.0 sec   833 KBytes  6.82 Mbits/sec   1.743 ms    0/  580 (0%)
[  3]  4.0- 5.0 sec   679 KBytes  5.56 Mbits/sec   2.463 ms    0/  473 (0%)
[  3]  5.0- 6.0 sec   591 KBytes  4.85 Mbits/sec   2.567 ms    0/  412 (0%)
[  3]  6.0- 7.0 sec   594 KBytes  4.87 Mbits/sec   2.928 ms    0/  414 (0%)
[  3]  7.0- 8.0 sec   596 KBytes  4.88 Mbits/sec   3.948 ms   31/  415 (7.5%)
[  3]  7.0- 8.0 sec  31 datagrams received out-of-order
[  3]  8.0- 9.0 sec   590 KBytes  4.83 Mbits/sec   6.348 ms    0/  411 (0%)
[  3]  9.0-10.0 sec   597 KBytes  4.89 Mbits/sec   2.316 ms    0/  416 (0%)
[  3] 10.0-11.0 sec   593 KBytes  4.86 Mbits/sec   6.129 ms    0/  413 (0%)
```

Fig. 11(a). Leitura do tráfego na porta 5000

```
rssouza@rssouza-pc:~$ iperf -s -u -p 5001 -i1
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[  3] local 192.168.20.125 port 5001 connected with 192.168.20.156 port 43610
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0- 1.0 sec   357 KBytes  2.93 Mbits/sec   7.306 ms    0/  249 (0%)
[  3]  1.0- 2.0 sec   355 KBytes  2.90 Mbits/sec   4.740 ms    0/  247 (0%)
[  3]  2.0- 3.0 sec   357 KBytes  2.93 Mbits/sec   6.293 ms    0/  249 (0%)
[  3]  3.0- 4.0 sec   356 KBytes  2.92 Mbits/sec   6.241 ms    0/  248 (0%)
[  3]  4.0- 5.0 sec   356 KBytes  2.92 Mbits/sec   6.616 ms    0/  248 (0%)
[  3]  5.0- 6.0 sec   356 KBytes  2.92 Mbits/sec   7.345 ms   31/  248 (12%)
[  3]  5.0- 6.0 sec  31 datagrams received out-of-order
[  3]  6.0- 7.0 sec   355 KBytes  2.90 Mbits/sec  10.004 ms    0/  247 (0%)
[  3]  7.0- 8.0 sec   357 KBytes  2.93 Mbits/sec   4.689 ms    0/  249 (0%)
```

Fig. 11(b). Leitura do tráfego na porta 5001

```
rssouza@rssouza-pc:~$ iperf -s -u -p 5002 -i1
------------------------------------------------------------
Server listening on UDP port 5002
Receiving 1470 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[  3] local 192.168.20.125 port 5002 connected with 192.168.20.156 port 38571
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0- 1.0 sec   237 KBytes  1.94 Mbits/sec   6.593 ms    0/  165 (0%)
[  3]  1.0- 2.0 sec   237 KBytes  1.94 Mbits/sec  11.423 ms    0/  165 (0%)
[  3]  2.0- 3.0 sec   237 KBytes  1.94 Mbits/sec   9.821 ms    0/  165 (0%)
[  3]  3.0- 4.0 sec   237 KBytes  1.94 Mbits/sec  298.748 ms    3/  168 (1.8%)
[  3]  3.0- 4.0 sec  28 datagrams received out-of-order
[  3]  4.0- 5.0 sec   237 KBytes  1.94 Mbits/sec   6.637 ms    0/  162 (0%)
[  3]  4.0- 5.0 sec  3 datagrams received out-of-order
[  3]  5.0- 6.0 sec   237 KBytes  1.94 Mbits/sec   7.176 ms    0/  165 (0%)
[  3]  6.0- 7.0 sec   237 KBytes  1.94 Mbits/sec   7.287 ms    0/  165 (0%)
```

Fig. 11(c). Leitura do tráfego na porta 5002

O tráfego recebido na porta 5000 mantém uma taxa média de 4,86Mbps por estar classificado como o de maior prioridade, além de possuir um peso maior na garantia de taxa. Na porta 5001, referente a segunda classe, o tráfego foi de aproximadamente 2,93Mbps. A terceira e menos prioritária se manteve com uma média de 1.94Mbps.

IX. Conclusão

Como foi possível observar ao longo deste trabalho o sistema operacional Linux, possui aplicações e ferramentas muito eficientes para o controle do tráfego recebido em uma rede TCP/IP podendo ser utilizado facilmente em um sistema embarcado para operar como um roteador, gateway ou ponto de acesso altamente configurável e estável.

Existem vários projetos de sistemas embarcados utilizando Linux para o gerenciamento e controle de redes, dentre estes se destaca o OpenWrt que possui suporte a várias plataformas e, pode até mesmo ser usado em vários dispositivos de redes substituindo a firmware de fábrica do mesmo.

Referências

[1] SANTOS, A., L.; Controle de Banda em Redes TCP/IP Utilizando o Linux. 2010.79p. Monografia (Especialista em Administração em Redes Linux) - Departamento de Ciências da Computação, Universidade Federal de Lavras. Minas Gerais. 2010.

[2] JUNIOR, C., CORRÊA, A., O.; Controle de Banda no Linux. Sistema de Informação & Gestão de Tecnologia. Belém. v.6, 2009.

[3] G. Lucian, Designing and Implementing Linux Firewall and QoS. Birmingham, UK: Packt Publishing, 2006, pp. 137–138.

[4] Kit de desenvolvimento FriendlyARM mini2440. Blog do Sério Prado. Disponível em: http: //sergioprado.org/kit-de-desenvolvimento-friendlyarm-mini2440/. Publicado em Acesso em Abril de 2012.

[5] Overview. Friendly Arm. Disponível em: http: //www.friendlyarm.net/dl.php?file=mini2440_overview.pdf. Acesso em: Abril de 2012.

[6] Mini 2440 Dimension. Friendly Arm. Disponível em: http ://www.friendlyarm.net/dl.php?file=mini2440_dimension.pdf. Acesso em: Abril de 2012.

[7] Mini 2440 Manual. Friendly Arm. Disponível em: http: //www.friendlyarm.net/dl.php?file=mini2440_manual.pdf. Acesso em: Abril de 2012

[8] Downloads. Friendly Arm. Disponível em: http ://www.friendlyarm.net/downloads. Acesso em: Abril de 2012

[9] Mini2440 -- Memory layout e bootloader. Blog do Sergio Prado. Disponível em: http: //sergioprado.org/mini2440-memory-layout-e-bootloader. Publicado em 17 de Agosto de 2010. Acesso em Abril de 2012.

[10] Como funciona um Sistema Operacional. Scribd. Disponível em:http://pt.scribd.com/doc/91751378/14/O-kernel-Linux. Acesso em Maio de 2012.

[11] G. Lucian, Designing and Implementing Linux Firewall and QoS. Birmingham, UK: Packt Publishing, 2006, pp. 63–86.

[12] Manpages. Linux Advanced Routing & Traffic Control. Disponível em : http://lartc.org/manpages/. Acesso em Abril de 2012.

Ricardo da Silva Souza nasceu em Baependi, MG, em 12 de setembro de 1986. Graduando em Engenharia da Computação pelo Inatel.

De junho a dezembro de 2011 estagiou no ICC com desenvolvimento de software para sistemas embarcados. De janeiro a maio de 2012 estagiou no CPqD atuando na área de Qualidade de Serviço em redes Adhoc. Atualmente é Analista de Desenvolvimento na Samsung, onde desenvolve soluções para dispositivos móveis.

Andrelise Rafael Gonçalves, nasceu em Santa Rita do Sapucaí em 9 de Abril de 1989. Graduanda em Engenharia da Computação pelo Inatel. Foi bolsista pelo Finatel no programa de Iniciação Científica no ano de 2010 com o projeto "Sistema Áudio-Braille para Deficientes Visuais". Concluiu os módulos I e II do curso preparatório CCNA-Cisco em 2010.

Desde agosto de 2011 é estagiária no CPqD na área de desenvolvimento de sistemas ópticos embarcados.

Caio César Hollerbach Gomes, nasceu em Passos, MG, em 14 de abril de 1989. Graduando em Engenharia da Computação pelo Inatel.

Atualmente é estagiário da empresa Hábeis Soluções onde faz pesquisa e desenvolvimento de hardware para produtos eletroeletrônico.

Bruno Reis Côrrea,nasceu em Ipatinga, MG, em 26 de novembro de 1986. Graduando em Engenharia da Computação pelo Inatel. De abril a março de 2012 foi bolsista da FAPEMIG pelo projeto IPTV: Protocolos Utilizados. Em julho de 2011 reafirmo seu interesse pela área de redes de computadores, fazendo o segundo modulo da CCNA-Cisco.

Atualmente é estagiário da empresa InterLink, onde desenvolve soluções para redes comerciais e residenciais.