A NOVEL PACKET QUEUING AND SCHEDULING ALGORITHM AND ITS LINK SHARING PERFORMANCE FOR HOME ROUTER

Jin Yan[1], Chen Xiaoyuan[2]

[1]College of Information Engineering, [2]College of Economics and Management

Zhejiang University of Technology

Hangzhou 310023, China

Email: jy@zjut.edu.cn

Submitted: Nov. 5, 2013 ***Accepted: Feb. 4, 2014*** ***Published: Mar. 1, 2014***

Abstract—A home router, or a home gateway, is the node that resides between a public network and a home network for computers to share Internet connections. To insert customized queuing and scheduling codes into the embedded Linux kernel is a way to make the Linux-based home routers support Quality of Service (QoS). It is desirable that the small Linux kernel adopts an efficient queuing and scheduling algorithm to support link sharing, priority and traffic shaping. The algorithm presented in this paper is based on the concept of hierarchical link-sharing and each class in the hierarchy is bound with a token bucket, which can shape the traffic of this class. The analysis and experiment results presented in the paper show that through the novel way of weighted fair token sharing, this proposed algorithm guarantees basic bandwidth service for each class and enables weighted fair sharing of excess bandwidth, with which home computers connected to the network share the Internet service evenly.

Index terms: Home gateway, home router, quality of service, embedded linux, token bucket.

I. INTRODUCTION

A home gateway is a node residing between a public network and a home network [1][2]. With the increased penetration of broadband and with the technological advances in home networking, there is a considerable market demand for the home gateway devices [3]. A set-top box (STB) or a home router can be a candidate for the home gateway. A home router, known as a kind of residential gateway [4] or home gateway also, is a small hardware device joining multiple networks [5][6][7]. The broadband router is a commercial name of the home router. Home routers are nowadays designed for convenience in setting up home networks, particularly for homes with high-speed cable modems or DSL Internet services. With using more computers, smart-phones [8] and other home network devices within a single home, home routers are widely used to allow more network devices to share Internet connections for homes or small offices. The characteristics of these home routers (home gateways) heavily influence the quality and performance of the Internet service that these residential users receive [9]. They may support Internet WAN connection, firewall, and network address transforming, virtual private network and so on. But, normally, they do not support fair allocation of bandwidth resources between the connections. In a usual home router, incoming packets are passed into a main processor and then fed into a switch for access sharing. Hereby the processor is expected to be able to control the traffic so that all home computers connected to the network can share the Internet service evenly.

An embedded Linux is expected to do extremely well in the digital consumer electronics area such as digital TV STBs, digital cameras and digital video recorders. It is used for software-based home gateways as well [10] [11]. The support of traffic control (TC) is available from Linux kernel version 2.1.90 and is more comprehensive in the more recent kernels. However, memory volume of a home router is very limited and can only spare a small portion for Linux system codes. Thus, the embedded Linux tends to use earlier kernel versions that are relatively smaller, or the kernel is greatly reduced and no longer supports TC. On the other hand, the TC methods implemented in Linux kernel does not always fit the

practical uses. Thus, it is meaningful to insert customized queuing and scheduling codes into the Linux kernel to support TC whereas keeping the kernel small.

Supporting TC is to find an applicable queuing and scheduling method to control the incoming and outgoing traffic. There exist some well-known queuing and scheduling algorithms supported in Linux, also called queuing disciplines, such as Token Bucket Filter (TBF), Priority, Stochastic Fair Queuing, and Class Based Queuing (CBQ) [12]. CBQ is a classical hierarchical packet scheduler supporting link sharing and priority. But CBQ does not support traffic shaping. A user system, e.g. home system, attached to QoS (quality of Service) supporting networks should perform traffic shaping for the outgoing traffic [13]. Thus, a customized queuing discipline that supports hierarchical link sharing, priority as well as traffic shaping is on demand. M. Bechler et al. presented a new traffic shaper (called Flow Based Queuing, FBQ) based on Class-Based QoS for Linux that aims at shaping aggregate traffic as well as individual flows within an aggregate[13]. However, both CBQ and FBQ are too complicated to be included in embedded systems. In some home gateway schemes, such as the one reported in [2], priority mechanism has been built at the main processor, but to our knowledge, none was found to support hierarchical link sharing, which enables each home computer to share the Internet service evenly.

The reminder of the paper is organized as follows. In section II we briefly describe the hardware architecture of a home router. Then, the concepts and knowledge of packet queuing and scheduling as well as traffic shaping are summarized in section III. Our queuing and scheduling scheme for this specific usage is presented in section IV. Section IV introduces the Linux networking subsystem and how to implement our own codes into this subsystem for Linux to support the customized TC. Section V discusses the experiment results. Finally in section VI some concluding remarks are given.

II. PRINCIPLE OF HOME ROUTERS

As shown in Fig. 1, a modern home router mainly consists of a computer CPU or controller, e.g. ARM® chip, a switching unit, public network interface and home network interface (wired or wireless), random access memory (RAM) and a modest amount of flash memory for

persistent storage. Switching unit really interconnects the CPU network interface at the home router side and home network interface as shown in Fig. 1 (b). The CPU can be a general-purpose processor directed by the system's embedded software to provide router services. The Linux operating system is employed and all traffic passes through the CPU for filtering and TC in our case.

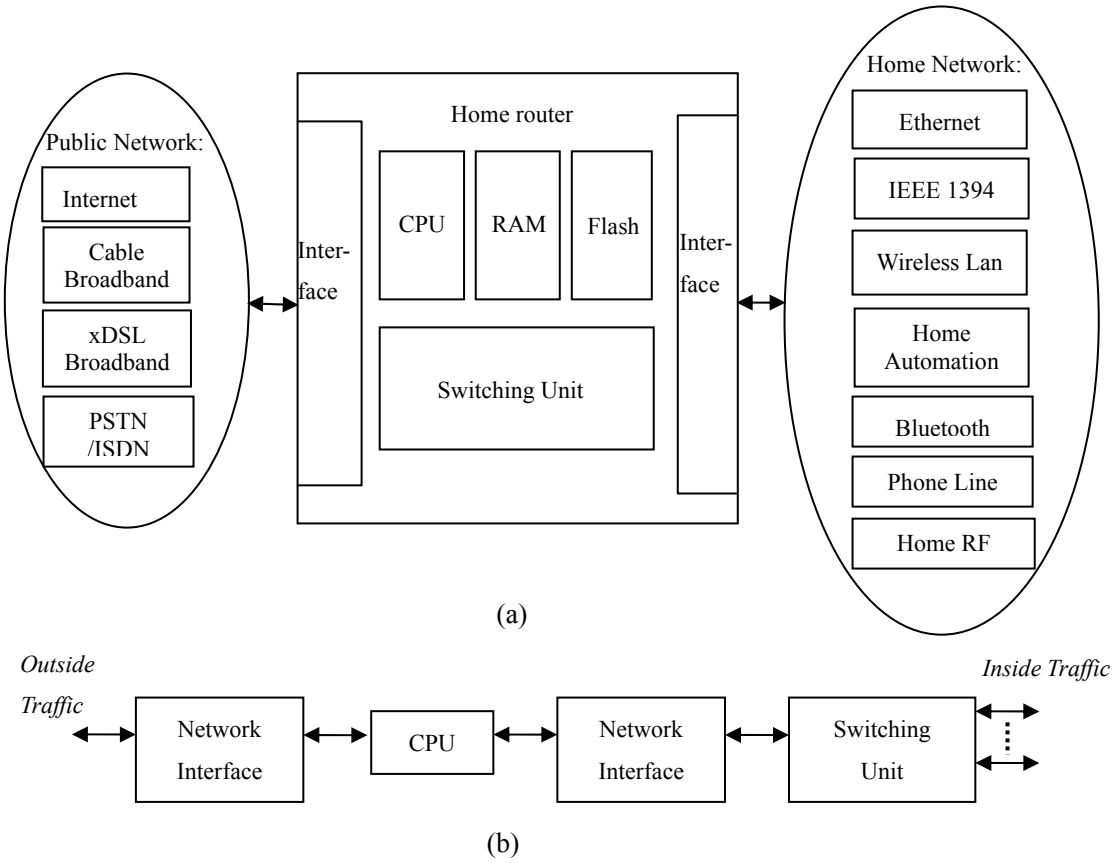


Figure 1. Hardware architecture of a home router
(a) functional blocks (b) traffic path model

III. PACKET QUEUING AND SCHEDULING OF HOME ROUTERS

Implementing a proper egress queue scheduling at the CPU of the home router is the key to QoS supporting. First in first out (FIFO) queuing is the most basic queue scheduling disciplines. In FIFO queuing, all packets are treated equally by placing them into a single queue, and then servicing them in the same order that they are placed into the queue. FIFO queuing is also referred to as first come first served (FCFS) queuing.

A FIFO queue is shown in Fig. 2. A simple FIFO queuing and scheduling method can accomplish traffic flow aggregation, but does not provide a mechanism to isolate the different levels and classes (types) of traffic flows, so each flow may suffer from the impact of other flows. Packets of unrelated flows share the same queue; one of those traffic flows with its volatility is likely to cause great impact on the each delay, jitter and packet loss performance of other flows. Certain types of traffic flow, such as TCP connection for e-mail, can tolerate delay but cannot tolerate packet loss, for this type of traffic a long queue is a better solution. For some other types of flows, such as the streaming video and audio, packets cannot stay too long in the network and can be dropped off. Therefore they are suitable to shorter queues. Unfortunately, FIFO queuing is not able to take care of such service differentiation.

Even if the traffic flows are of the same type, such as in the Internet connection sharing case discussed in this paper, FIFO queuing is not able to support service isolation and fair allocation of bandwidth in a crowded scenario. The flow with more packets and/or longer packet length will receive more bandwidth when sharing the output link. To support fair allocation of bandwidth and service differentiation, the QoS queuing and scheduling approach is necessary. The existing QoS queuing and scheduling approaches can be divided into two categories: Single level and hierarchical.

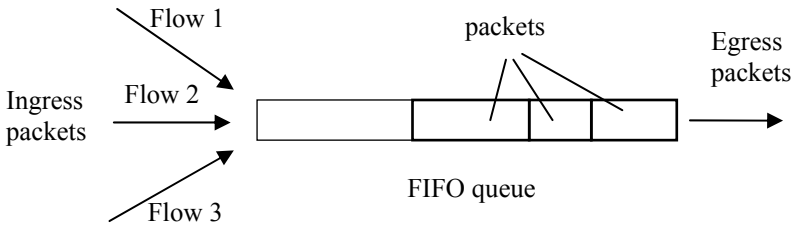


Figure 2. FIFO queuing

a. Single Level Queuing and Scheduling

Fig. 3 shows the typical model of single level queuing and scheduling. There are multiple FIFO queues to buffer the packets from different flows or different types of aggregations.

Strict priority scheduling method adopts a number of decreasing priorities for these FIFO queues. Only a given priority queue can be served when all higher priority queues are empty.

For example, Queue 1 has a higher priority than Queue 2, Queue 2 has a higher priority than Queue 3. As long as the link can transfer packets, the queue gets link service as quickly as possible. Queue 1 is served before considering Queue 2. Similarly, only when Queue 1 and Queue 2 are empty, Queue 3 gets service at the link rate.

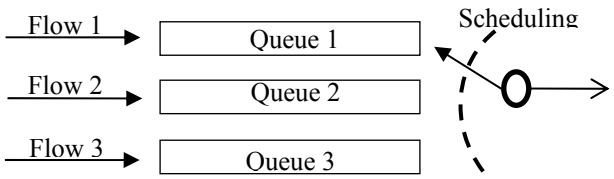


Figure 3. Single level queuing and scheduling

Priority scheduling supports service differentiation and it is able to provide low-latency service for delay sensitive traffic. However, bandwidth management capacity of priority scheduling is limited. It may starve low-priority queues on bandwidth demand if the higher priority queues are continuously backlogged.

Cyclic (Round-Robin, RR) scheduling method is to serve each queue each time in the move to transfer a packet so as to avoid any queue starvation. Empty queues are skipped. If each queue has a packet to send, each queue transfers one packet per cycle. If some queues are empty, then the other queues are more frequently served. The disadvantage of RR scheduling is that it can not manage delay, and its bandwidth allocation is not fair, too. The traffic flow of greater packets accounts for more bandwidth.

Difference Round-Robin scheduling (DRR) is the expansion of original Round-Robin scheduling. DRR records the number of bytes sent from a given queue, and this number is compared with the number of bytes that should be sent and the difference is as a "balance". The "balance" has been used for any given queue, modifying the queue service interval. Therefore, it can be adjusted the long-term bit-rate for each queue.

Another improvement to the RR scheduling is to recalculate the scheduling order constantly and the next served queue is selected according to the long-term average bandwidth assignment. Fair queuing (FQ) is to determine the next served queue according to fair bandwidth allocation among all queues. Weighted fair queuing (WFQ) is the improvement of FQ. It allows the allocation of different weight to a single queue. There are other scheduling

methods, such as starting time fair queuing (SFQ) and the worst-case weighted fair queuing (W^2FQ), etc, which are basically the improvements of scheduling methods described above.

WFQ is also an approximation of Generalized Processor Sharing (GPS). GPS is the ideal single-level scheduling scheme. Actual single-level scheduling schemes use GPS as an approaching object, and GPS itself cannot be achieved in engineering, which is because it requires that the scheduling unit is infinitely small, but actually scheduling unit is a single packet. However, GPS can be used as the general theoretical model for evaluating a single-stage scheduling scheme, it reflects the general nature of a single-stage scheduling,

1) Latency is limited.

2) Work conserving, i.e., as long as there are packets waiting at any queue, the service never ceases and then, the bandwidth capacity is never wasted.

3) Bandwidth is equally or by the weights assigned to different queues.

Let $W_i(t_1, t_2)$ be amount of traffic served in the interval $[t_1, t_2]$ for queue i, $W(t_1, t_2)$ be total amount of service provided by the server, transferring packets at outgoing link rate, during the same period, ϕ_i be the service share (or weight) of queue i.

A single-level GPS scheduler is defined as one for that,

$$W_i(t_1,t_2) = \frac{\varphi_i}{\sum_{j \in S(t_1)} \varphi_j} W(t_1,t_2) \tag{1}$$

holds for any interval $[t_1,t_2]$ during which $S(t_1)$, the set of backlogged queues (i.e., queues with packets to be sent) at time t_1, does not change. Let S be the set of all queues, then

$$W_i(t_1,t_2) \geq \frac{\varphi_i}{\sum_{j \in S} \varphi_j} W(t_1,t_2) \tag{2}$$

holds for any interval $[t_1, t_2]$ during which queue i is continuously backlogged. This formula indicates that each queue receives a guaranteed minimum share of the outgoing link capacity during any of its backlogged periods, regardless of the behaviors of other queues.

GPS can provide a delay bound to a backlogged queue, with an average rate not greater than its minimum guaranteed rate:

$$D_i \leq MTU_i / r_i \tag{3}$$

where D_i is traffic delay of queue i, r_i ($=\phi_i \cdot r$) is its minimum guaranteed rate and r is the

server's rate (link rate). MTU_i is the maximum amount of traffic backlogged in queue *i*.

b. Hierarchical Queuing and Scheduling

The single-level scheduling applies unified bandwidth management. Scheduling can be hierarchical for the cases that queues have a hierarchical relationship. If two users share a link and each of them wants QoS control of its traffic, the bandwidth isolation between two users and further packet scheduling for managing traffic transportation are in demand. Any single-level scheduling method cannot be a solution to a hierarchical link-sharing problem.

In hierarchical link-sharing, there is a class hierarchy associated with each link specifying the resource allocation policy for the link. A class represents a traffic flow or some aggregations of traffic flows grouped according to administrative affiliation, protocol, traffic type, or the other criteria. Fig. 4 shows an example of three level class hierarchies for a physical link that is shared by three virtual links, Premium Service Link (PSL), Assured Service Link (ASL) and Best Effort Service Link (BESL), corresponding to the three differentiated services specified by the Internet Society [14]. Below each of the three classes, there are offspring (child) classes grouped with different network users. In this example there are only three users, A, B and C. Each child class is associated with its resource requirements. A rated bandwidth is the minimum amount of service that the class should receive when there is enough demand. In Fig. 4, Link represents the physical link between a user system and the outside network, and Class PSL, ASL and BESL represent Premium, Assured and Best Effort Service traffic aggregations, respectively. For efficiency, a user should be allowed to borrow bandwidth from others that is not presently in use. If these users are from one group that does not allow borrowing bandwidth from other groups, one should insert another level just above the leaf classes, classes that have no offspring, to clarify these different groups.

Of all hierarchical queuing and scheduling methods, Class-based queuing (CBQ) is a well accepted one that allows traffic to share bandwidth equally after being grouped by classes [12]. CBQ can control the amount of bandwidth which each user gets, divide traffic of each user into different classes and guarantee each class a minimum bandwidth. CBQ can also assign priority to each class. This will enable real-time or delay sensitive applications such as voice and video. This can also be the first to get idle bandwidth on the link, thereby reducing

the delay and packet loss so that the QoS can be guaranteed. Though CBQ provide bandwidth isolation between classes, traffic of a class may borrow the idle bandwidth from the other classes when it bursts and needs more bandwidth than the guaranteed, thereby increasing efficiency of bandwidth utilization.

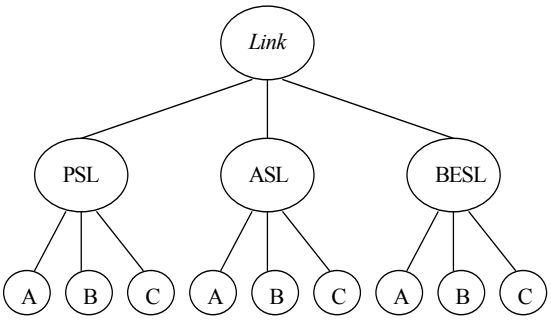


Figure 4. The class hierarchy

CBQ solves problem of link sharing, but handles real-time classes by the allocation of spare bandwidth according to priority, namely, delay distribution is achieved through allocation of bandwidth. Hierarchical Fair Service Curve (HFSC) can allocate delay and bandwidth, respectively, based on so called traffic service curve concept, by distribution of service curves, the allocation of delay and bandwidth is achieved [15].

A hierarchical GPS (H-GPS) scheduler is a hierarchical integration of one-level GPS schedulers. The root node corresponds to the physical link and each leaf node corresponds to a flows or one type of aggregation.

The class hierarchy is the integration of a basic tree with one parent node (class) and its child node. A H-GPS scheduler is defined as one for which formula (1) also holds here but the variable definitions are different. Now $W_i(t_1, t_2)$ is the amount of traffic served in the interval $[t_1, t_2]$ for the child node i, $W(t_1, t_2)$ is the total amount of service provided by the parent node during the same period, ϕ_i is the service share of child node i, and $S(t_1)$ is the set of backlogged child nodes. Similarly, formula (2) also holds here but the variable definitions are different and S is the set of all child nodes.

c. Traffic Shaping

Multimedia communication terminals generate traffic flow that is not necessarily adaptive to the service requirements, such as rate requirement. If no meeting these requirements, QoS

cannot be guaranteed. Usually, video and Internet traffic has burst rate much higher than average rate. Setting the service rate at burst rate is too wasteful, so traffic should be shaped, made smooth, prior to entering the network, traffic shaping is used to control and limit the average rate and burst size.

Token Bucket (TB) is widely used as a traffic shaper. Fig. 5 shows the Token Bucket model. Operation of TB is as follows [16]:

Data flows into bucket of TB from the left in quanta called packets. Tokens flow into the bucket of TB from the top at rate *r*. When the bucket with the depth *b* is full of tokens, new tokens are thrown away. Each token is worth a defined number of bytes. It is easier if we think of each token as being worth one byte. When a packet arrives from the left, if there are a number of tokens in the bucket at least equal to the number of bytes in the packet, the packet may exit the system immediately. If there are not enough tokens in the bucket, the packet can be buffered and not released until a sufficient number of tokens arrive in the bucket.

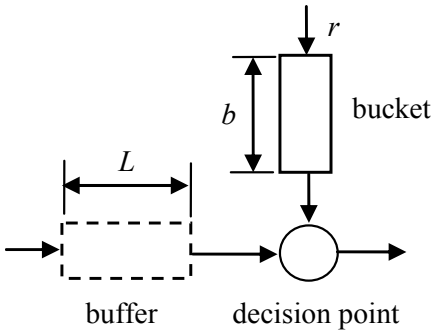


Figure 5. Token bucket

Traffic shaping is helpful to reduce chance of congestion. For upstream transporting, it helps the home router to send well-formed traffic that will not possibly be rejected or penalized at the network border. In most QoS supporting network architectures some sort of agreement about the kind of service exists between the network and the end system[12]. In case of ATM this is called a traffic contract, in case of the Differentiated Services architecture similar issues are handled in service level agreement (SLA) and traffic conditioning agreement (TCA). In general, these agreements define regulated traffic the network accepts from the end system. If

packets are sent from the end system into the network that violates the agreement, the forwarding may be denied or at least not guaranteed depending on the specifications.

Existing queuing and scheduling methods, whether they are hierarchical or single-level, do not support traffic shaping directly. Flow Based Queuing (FBQ) [10] can support traffic shaping directly as well as hierarchical link sharing on per flow basis. But it is too complicated to be applied in this kind of home router. On the other hand, FBQ is not flexible enough to set the traffic shaping classes selectively.

In this particular case of Fig. 4, traffic of PSL and ASL should be shaped to guarantee QoS. Traffic from different users, however, needs not be shaped personally.

IV. NOVEL PACKET QUEUING AND SCHEDULING ALGORITHM AND ITS LINK SHARING PERFORMANCE ANALYSIS

a. Our Queueing and Scheduling Algorithm for hierarchical Link Sharing

Fig. 6 shows the architecture of scheduler that matches the class hierarchy, shown in Fig. 4. Traffic from each class is controlled by the Token Buckets (TBs). Only TBs of leaf classes have buffers. Then in Fig. 6, packets can be en-queued in TB_{2j} (j=1,…,9) and de-queued to the upstream TB, the *Link*. When the packet at the head of a leaf class buffer is eligible to depart from the present TB (accumulating enough tokens), it passes to the upstream TB. For example, when a packet at TB_{21} is eligible to depart, it passes to TB_{11}. It then passes to *Link* if TB_{11} lets it go, or it remains at the head of the buffer of TB_{21} if TB_{11} has not accumulated enough tokens yet. As long as a packet at TB_{21} is eligible to depart, one says that it has passed to TB_{11} whether it remains in TB_{21} buffer or not. If TB_{11} accumulates enough tokens, then the packet passes to the upstream class or *Link*.

If the packet at a different sibling TB, e.g. TB_{22}, is eligible to depart simultaneously, an arbitrator is needed to determine which packet goes first. There are arbitrators at the traffic converging points. These arbitrators may be the RR schedulers or priority schedulers. Both of schedulers are simple. In Fig. 6, S_{2j} (j=1, 2, 3) uses RR schedulers, whereas S_{11} uses a strict priority scheduler, TB_{11} possesses the highest priority and TB_{13} possesses the lowest one. Thus, if *Link* is empty, packets from TB_{11} go to *Link* first, packets from TB_{12} may go to *Link*

only if there is no eligible packet from TB_{11}. Packets from TB_{13} may go to *Link* only if there is no eligible packet from TB_{11} and TB_{12}. In this way, the packets for Premium Service undergo the least delay.

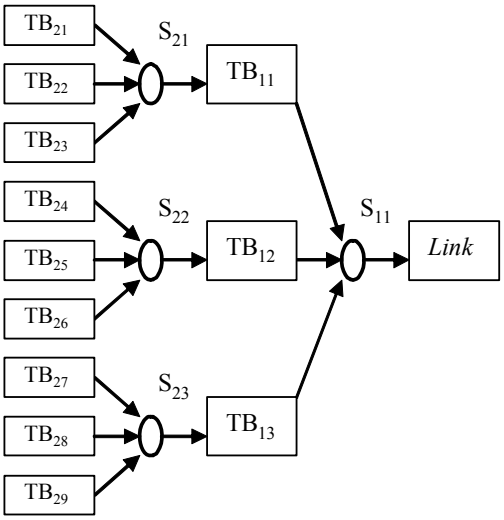


Figure 6. Architecture of the scheduler

Assume class i is bound with TB_i with the average rate r_i and bucket depth b_i. The head of line (HOL, at the head of the queue) packet with the size of $packetsize_i$ becomes eligible to pass into its parental class at time t_2 if the following condition is satisfied,

$$packetsize_i \le T_i(t_2) \tag{4}$$

where $T_i(t_2)$ is the tokens the packet class gets at the time of t_2. The algorithm is cyclic (periodic). For each cycle, denoting beginning time t_1 and ending time t_2, then

$$T_i(t_2) = \min[b_i, T_i(t_1) + r_i(t_2 - t_1)] \tag{5}$$

When the HOL packet passes away, just virtually, maybe still remains at the leaf class queue, $T_i(t_1)$ is updated with:

$$T_i(t_1) = T_i(t_2) - packetsize_i \tag{6}$$

The packet can really be dequeued from the leaf class queue to the physical link after all the ancestral classes permit it to be dequeued. Obviously, each class can perform traffic shaping, but only a few classes are really needed. In the example above, only PSL and ASL are really needed to perform the traffic shaping. Restrained by the TBs, a class cannot receive excess bandwidth from its parental class when some sibling classes are idle with no packet for

scheduling. This will result in inefficient hierarchical packet scheduling. To make the most use of bandwidth capacity, one hope that child classes with packets backlogged and without traffic shaping requirements can always deliver the packet to their parental class once the parental bandwidth capacity allows. The above TB algorithm is thus revised. For each packet at child classes, the anticipated departure time can be calculated according to (4) and (5). Let $\Delta t=t_2-t_{\min}$, where $t_{\min}$ is the earliest anticipated departure time for a packet of all child classes,

$$t_{\min}(t_2) = \min\{[packetsize_i - T_i(t_1)]/r_i\} \tag{7}$$

The packet with $t_{\min}$ is then eligible to leave for the parental class. Tokens of the class that the departed packet belongs to is set to zero, tokens of the other classes are calculated for the purpose of fair allocation of tokens,

$$T_i(t_2) = \min[b_i, T_i(t_1) + r_i(t_2 - t_1 + \Delta t)] \tag{8}$$

The combination of N sibling classes with a parental class is the basic architecture of the hierarchy. Link sharing function is an important one that our algorithm is going to be realized. For application such as home routers, only a basic architecture of the hierarchy is sufficient to support fair allocation of bandwidth resources. In addition, link-sharing performance of the hierarchy can be analyzed from the basic architecture. Therefore, in the next analysis, we focus on basic architecture.

b. Link Sharing Performance Analysis

Let $P_i(t_1, t_2)$ be the tokens that class i receives in the interval $[t_1, t_2]$, ϕ_i be the service share of class i, r_i and b_i be the average rate and bucket depth of TB_i (i=1,…,N), and r and b be the average rate and bucket depth of their common parental class, respectively. $r \geq \sum_i r_i$, $r_i=\phi_i \cdot r$, $\sum \varphi_i \leq 1$. It holds for each child class,

$$P_i(t_1, t_2) = r_i(t_2 - t_1 + \Delta t) = \varphi_i r(t_2 - t_1) + \varphi_i r \Delta t, \quad \forall i \in B(t_1) \tag{9}$$

where for any interval $[t_1, t_2]$ during which $B(t_1)$, the set of backlogged, having one or more packets waiting to pass, TB_i at time t_1, does not change, and the backlogged TB_i is continuously backlogged.

It is noted that $\Delta t \geq 0$, from (9), one finds that class *i* receives the tokens no less than that its parental class gives at the weight of ϕ_i during $[t_1, t_2]$, and it receives excess tokens at the weight of ϕ_i too if there is redundant bandwidth. The way of scheduling tokens is in accordance with the concepts of H-GPS. The link-sharing goal of weighted fair bandwidth allocation can be achieved based on weighted fair token sharing.

Considering the basic architecture of the hierarchy, one knows that the leaf class *i* can get no less than $r_i (t_2 - t_1)$ tokens during $[t_1, t_2]$. For *i*-th flow constrained by a leaky bucket (σ_i, ρ_i), the traffic with no more than $\sigma_i + \rho_i(t_2 - t_1)$ will enter class *i*. Provided that $r_i \leq \rho_i$, all the traffic can pass to the parental class if extra time σ_i / r_i , at most, is spent. The traffic with the amount of $\sum_i [\sigma_i + \rho_i (t_2 - t_1)]$ may enter the parental class during $[t_1, t_2]$ whether the arbitrator uses RR or Priority mechanism. Let *r* be the average rate of the parental class, normally, $r \geq \sum_i r_i$. No less than $r(t_2 - t_1)$ tokens are accumulated at the parental class TB during $[t_1, t_2]$. Thus, no more than the time $(\sum \sigma_i)/r$ is additionally needed for all the traffic to pass to the parental class. It is noted that the parental class and child classes accumulate tokens in parallel way. It can be concluded that for any interval $[t_1, t_2]$, the maximum delay a passing traffic may undergo is

$$d_m = \max\{\sum \sigma_i / r, \max(\sigma_i / r_i)\} \tag{10}$$

Therefore, the delay is bounded. If for each child class *i,* all σ_i are equal and all r_i are equal, then (10) is reduced to

$$d_m = \sigma_i / r_i \tag{11}$$

V. IMPLEMENTATION OF ALGORITHM IN EMBEDDED LINUX

The core architecture of the home router is an embedded computer system. The demand for Linux in embedded systems has been rapidly growing over the last decade, making Linux the primary platform for embedded development across a variety of systems and uses, including home routers.

Linux is a free Unix-type operating system originally created by Linus Torvalds with the assistance of developers in the world. Developed under the GNU General Public License, the source code for Linux is freely available to everyone. Linux is easier and more flexible to install and administer than UNIX. But Linux has many of the same commands and programming interfaces as traditional UNIX. Although most Linux systems run on PC platforms, Linux can also be a reliable workhorse for embedded systems. A fully featured Linux kernel requires about 1 MB memory. However, Linux micro-kernel actually consumes very little of memory. With networking stack and basic utilities, a complete Linux system runs quite nicely in 500 KB memory on an Intel 386 microprocessor, with an 8-bit bus.

In this section we shall describe how to implement packet queuing and scheduling algorithm inside Linux kernel to support traffic control for QoS delivery. At first, we should know the Linux networking subsystem.

a. Linux Networking Subsystem

Fig.7 describes the Linux networking subsystem structure that achieves layered structure of TCP/IP protocol family. Socket module provides the interface for network application programs to use the TCP, UDP, or direct IP, even raw Ethernet services.

The network device layer below the IP layer of network may include Ethernet or other networking devices. Network devices appear in the /dev directory of Linux file system only after the low-level discovery and initialization of these devices. ARP (address resolution protocol) provides address resolution functionality. It is between the IP layer and network device layer.

For the actual implementation of network protocols, such a hierarchical structure has brought a number of issues, including much obvious one, the data should be passed between different protocol modules, because each layer must find their own protocol-specific header and trailer, resulting in frequent transfer of data and more data buffers. One way to solve the problem is to copy data between the buffers, but this method is less efficient. Linux uses socket buffer (*sk_buff*) for transferring data between different modules. *sk_buff* contains some pointers and length of the information, shown in Fig. 8. Different modules can use the same structure for data processing.

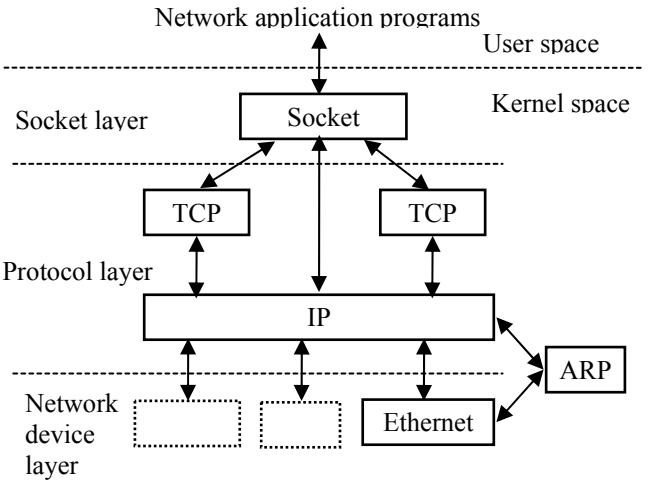


Figure 7. Linux networking subsystem structure

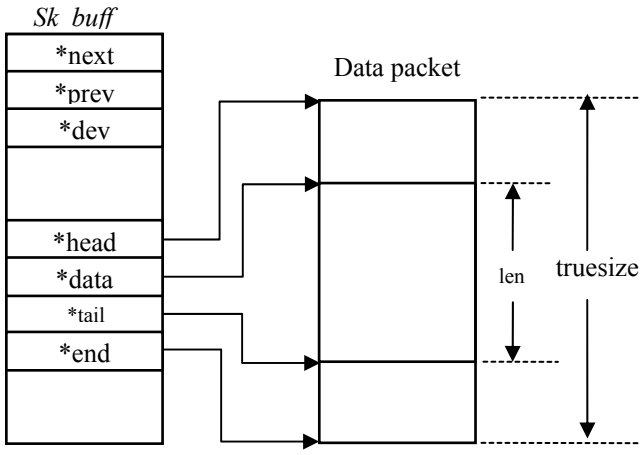


Figure 8. Data structure of *sk_buff*

The *len* and *truesize* describe the current protocol data packet length and the actual length of the data buffer. *sk_buff* handling codes provide some standard functions, these functions are used for application to add or remove protocol headers and tails. Using these functions can safely manipulate *sk_buff* structure.

When the network device receives the data packet from the network, the received data should be converted into the *sk_buff* structure and added to *backlog* queue. When the *backlog* queue becomes full, the received *sk_buff* data will be discarded. When a new *sk_buff* is added to the *backlog* queue, the network *bottom half* handling process is marked ready, allowing the

process scheduler to trigger this *bottom half* handling routine, sending the packet to a proper layer.

Data packets can be generated not only by application layer of computer but by the network incoming as well. No matter how the data is generated, the network handling code must establish *sk_buff* structure to contain the data, while the data is passed between protocol layers, a different header and protocol tail should be added.

The *sk_buff* should be passed to a network device before it can be transmitted. For each of the IP data packets to be transmitted, IP routing table is used to resolve target IP address of the route. For each target IP address that can be routed from the routing table. The routing table can be asked to return a route that is described in the *rtable* data structure. This data structure includes source IP address, network device, device data structure, pre-built hardware address and header information. Hardware header information is related to the network device, containing the physical address of the source and destination as well as other media information. For the Ethernet networking, the source and destination address is the physical address of the Ethernet card. The physical address information in the header sometimes needs to use the ARP protocol to get.

b. Linux Routing

Linux routing function is mainly implemented in the kernel. Its function block diagram is shown in Fig. 9. When a packet arrivals, the kernel checks IP address. The packet is sent to the high-level (Upper layers) if the IP address on the packet is the local address, or the packet is forwarded to the output queue if the IP address is the non-local. The current Linux kernel supports TC, the output queue can select queuing and scheduling mechanisms implemented in the Linux kernel.

From the kernel version 2.1.105 Linux began supporting QoS, i.e., the queuing and scheduling mechanisms. It can set up TC to networking devices such as network card. The pointer to the output queue is saved in the device data structure (see the kernel code netdevice.h), so that the device and some kind of QoS mechanism can be linked together. The IP layer adds a MAC header to the passing packet, call *dev_queue_xmit( )* (see the kernel code net / core / dev.c) to submit this packet to the device driver. The function is a piece of codes

shown as follows, in which *skb* is the kind of *sk_buff* data structure and actually is the packet to be sent,

```
……
q=dev->qdisc;
if(q->enqueue){
   q->enqueue(skb, q);
   qdisc_wakeup(dev);
return 0;
}
……
if(dev->hard_start_xmit(skb, dev)==0)
……
```

The codes from the second line to sixth line perform packet enqueuing and dequeuing, and then execution of *dev->hard_start_xmit(skb, dev)* to send the packet to the selected network device. We tried to add codes of QoS scheduling just before the above piece of codes. Our tests show that a stable new kernel can be achieved after such a modification.

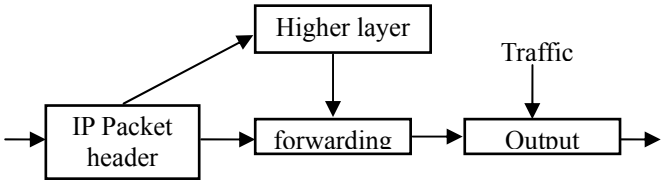


Figure 9. Linux routing function blocks

c. Our Implementation

To accomplish our own QoS scheduling method for TC, some specific code modules should be inserted into the Linux network subsystem. These modules include functions of entering, leaving, re-entering, and droping from the queue. They are described as follows:

1) enqueue_QoS( )

This function is designed to put the incoming packet into the proper queue. Firstly, the packet is classified in accordance with the information contained in the UDP or TCP overhead. IP

protocol uses IP addresses to deliver the data to the correct computer, TCP and UDP protocol uses port number (PORT) to deliver data to the correct application.

TCP and UDP have their own 16 bit port numbers, namely, providing 216 different ports. These ports are divided into two parts, the port numbers less than 256 are the system reserved ports for the global allocations, and the other numbers are ports freely allocated in a local manner. Here are some of the standard TCP port numbers,

Protocol	port number
http	80
ftp	21
telnet	23
pops	110
smtp	25

The same type of multimedia streams can be allocated a fixed local port to facilitate identification. The different flows can be distinguished according to the port information and IP address. As the packet *skb* has also included the MAC header in before QoS Scheduling, MAC header information can be used for classifying different flows, too.

As for the MAC header of an Ethernet frame, the preamble is not included in *skb*, thus, the hardware destination address is stored at *skb->data* to *skb->data+5* and source address is stored at *skb->data+6* to *skb->data+11*. Considering IP address, the destination address is stored at *skb->data+25* to *skb->data+28* and source address is stored at *skb->data+29* to *skb->data+32*. The locations of TCP and UDP port number in *skb* are the same, the destination port is stored at *skb->data+37* to *skb->data+38* and source port is stored at *skb->data+39* to *skb->data+40*.

2) dequeue_QoS

This function calls in the scheduling function and returns a pointer to the packet that can be de-queued. If there is no packet can be de-queued, it returns a pointer pointing to NULL. The *sk_buff* structure contains pointers for constructing a two-way link table, the kernel code has a series of functions regarding *sk_buff* queue operations, and one can use them to constitute queues. But it is easier to keep stability of the modified kernel by establishing our own set of queue manipulation functions. The queue data structure is defined as SEQUEUE:

```
typedef struct{
    struct sk_buff *elements[MLEN];
    int front,rear;
} SEQUEUE;
```

where MLEN is the maximum length for the queue. The established queue manipulation functions are listed below:

*init_sequeue(SEQUEUE *Q)* : initialize the queue named Q;

*add_sequeue(SEQUEUE *Q,struct sk_buff *x)* : put a packet into the FIFO queue Q;

*delete_sequeue(SEQUEUE *Q,struct sk_buff *x)* : dequeue a packet;

*front_sequeue(SEQUEUE *Q,struct sk_buff *x)* : read the packet at the front of the queue;

*empty_sequeue(SEQUEUE *Q)* : determine whether the queue is empty or not.

3) requeue_QoS( )

After leaving the queue, the packet is finally sent to the device driver and then to the hardware by calling *hard_start_xmit ( )* function. The device driver may refuse to accept the packet due to congestion or other reasons, then *hard_start_xmit ( )* function call fails. In this case, the packet should be put back into the queue and should stay at the head of line, waiting for the next chance to be sent.

4) drop_QoS( )

When a packet arrives, if the queue it belongs to is full or no spare queue to enter, this packet will be dropped. dropping mainly occurs at the course of en-queuing, so the function *drop_QoS ( )* is usually called by *enqueue_QoS( )*.

The complete QoS packet scheduling process is shown in Fig.10.

The packet receiving and forwarding process and related functions are shown in Fig. 11. When data packet arrives at the network port, the network interface circuit issues an interruption request, the computer CPU responds to the request by setting up a sign bit at *bh_active* field and returns from interruption swiftly. The *bottom half* handling routine will deal with it afterwards. The Linux kernel scheduler performs the *bottom half* handling prior to scheduling for execution of a process. The *bottom half* handling routine calls *ip_rcp( )* to receive packet, checking whether the packet is a legitimate IP packet or not. The packet will be discarded if it is illegal, and if it is legal *ip_rcv_finish( )* will be called. Inside

ip_rcv_finish( ), *ip_route_input( )* is called to determine whether the packet is sent to the high layer of this host or needs forwarding, according to source IP address, target IP address and routing table. By setting the routing table, the system can be configured to become a router or gateway, or even just for high-level "filtering" to certain IP addresses. When the packet is destined to this host, *ip_local_deliver( )* is called to sent it to the high-level. If the packet should be sent to other hosts, *ip_forward( )* is called. *ip_forward( )* continues to call *ip_forward_finish( )*, and further call *ip_send( )* to determine whether the packet is too long. If being too long, it calls *ip_fragment( )* for packet fragmentation, and then calls *ip_finish_output( )* to continue to send the packet. *ip_finish_output( )* determines the right network port to send the packet and calls *ip_finish_output2( )* to add the MAC header by calling *neigh_resolve_output( )* inside it, and then, calls *neigh_connected_output( )*, which calls *dev_queue_xmit( )* to send the packet to our own QoS scheduler. The scheduler consists of *enqueue_QoS( )*, *dequeue_QoS( )*, *requeue_QoS( )*, and *drop_QoS( )*, and finally calls *hard_start_xmit( )* to send the packet to the proper network port.

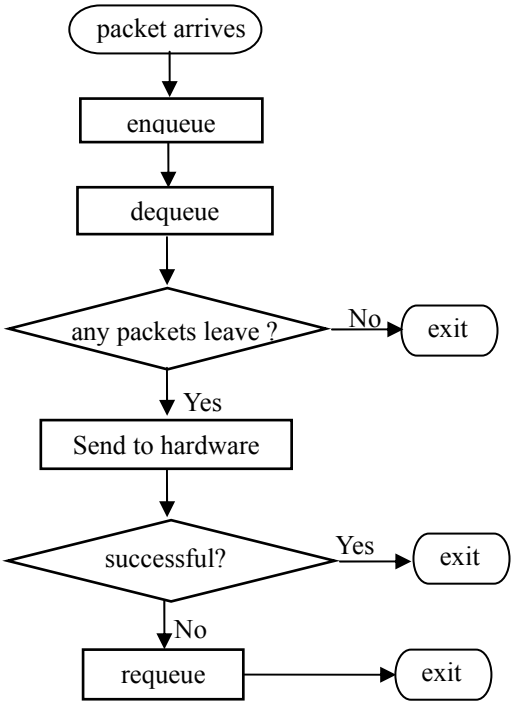


Figure 10. QoS packet scheduling process

In Fig. 11, the functions in italics are the customized queuing and scheduling codes, whereas other functions are inherent in Linux kernel. If the system is not only for packet scheduling, but also for protocol conversion and other functions, such as filtering, the received packet should be transferred to high layer by *ip_local_deliver_finish( )*. And *ip_queue_xmit( )* is responsible for routing the packet from high layer process, adding IP header, and then calling the *ip_queue_xmit2( )* to continue sending the packet. *ip_queue_xmit2( )* determines whether the packet is too long, calling *ip_fragment( )* to make segmentation if it is, and then calls *ip_output( )*, which will call *ip_finish_output( )*. The afterward process is the same as above.

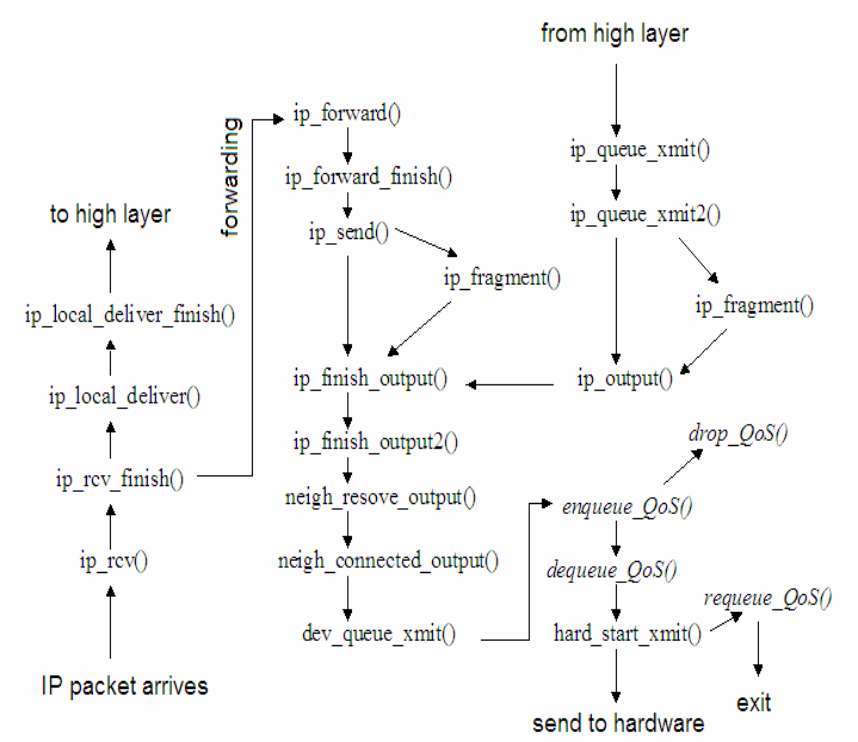


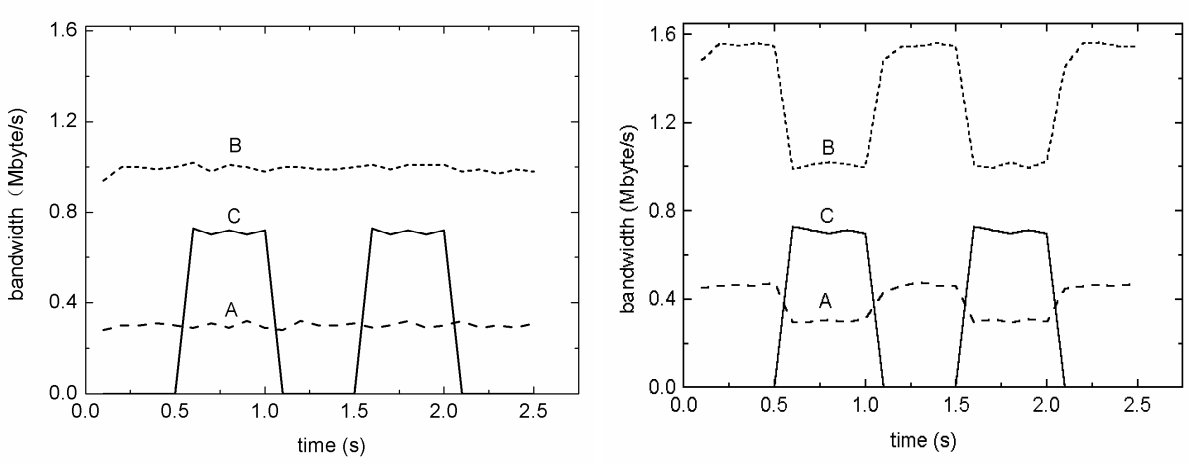
Figure 11. Packet receiving and forwarding process inside the Linux network subsystem

VI. EXPERIMENT RESULT

An experiment was made on a computer with Linux operating system installed to evaluate the link-sharing performance of the algorithm. Customized queuing and scheduling codes were inserted into the Linux kernel codes at net/core/dev.c. The hierarchical scheduler shown in Fig. 4 and Fig. 6 were programmed and the basic architecture of the hierarchy was focused on. Assuming that PSL had bandwidth reservation of 2 M-byte/s and could not exceed 2 M-byte/s,

the three classes at level two, i.e. A, B, and C, had bandwidth reservations of 0.3, 1.0, 0.7 M-byte/s, respectively, and they were allowed to borrow bandwidth from their sibling classes, all flows were continuously backlogged except for the 0.7 M-byte/s flow which was an ON–OFF source. Bandwidth reservations were accomplished by setting the average rates of TBs. The traffic load was generated by a self-timed user-level program, which sent raw packets of size 1500 bytes for each flow at the required rates. Packets were sent into the scheduler by socket and looped back to the user-level program through lo (the loop-back device in Linux). The user-level program measured the packet rates for different flows.

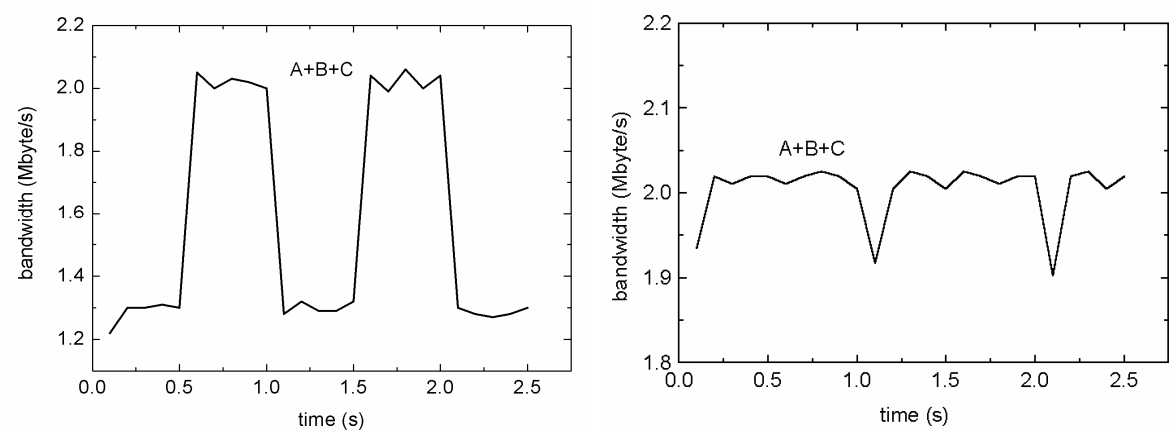
Fig. 12(a)(b) shows the bandwidth versus time graph for three flows at level 2 in the hierarchy in a) and b) cases. To compute the bandwidth, a 0.1-s averaging interval was used for all flows. As can be seen, if not allowed to borrow bandwidth, classes A and B received their guaranteed rates, i.e., 0.3 and 1.0 Mbyte/s whether the ON–OFF flow was active or not. If allowed to borrow the bandwidth, classes A and B received 0.46 and 1.54 M-byte/s bandwidth allocations, respectively, i.e., they received additional bandwidth 0.16 and 0.54 M-byte/s when the 0.7-Mbyte/s ON–OFF flow was inactive. The parental class PSL had an excessive bandwidth 0.7 M-byte/s after satisfying the bandwidth reservations of classes A and B. It allocated the excessive bandwidth among its active child classes, A and B, according to their shares, i.e., $0.16/0.54 \approx 0.3/1.0$. When the ON–OFF flow was active, they all received their guaranteed rates, i.e., 0.3, 1.0 and 0.7 M-byte/s, respectively.



(a) Not allowed to borrow bandwidth (b) Allowed to borrow bandwidth

Figure 12. The bandwidth versus time graph for three flows

Fig.13 (a)(b) shows the bandwidth versus time graph for the sum of service three flows received in the two cases. If not allowed to borrow the bandwidth, PSL is under loaded when the ON–OFF flow was inactive. If allowed to borrow the bandwidth, the sum approaches PSL bandwidth reservation, though the curve has downward thorns at 1 and 2 second, when the ON–OFF flow was inactive. These downward thorns tell us that excessive bandwidth allocation costs a small portion of time.



(a) Not allowed to borrow bandwidth (b) Allowed to borrow bandwidth

Figure 13. The bandwidth versus time graph for the sum of services

Ⅶ. CONCLUSIONS

Inserting the customized queuing and scheduling codes into the embedded Linux kernel is a way to enhance the Linux-based home routers for Quality of Service. The small Linux kernel can adopt a queuing and scheduling algorithm to support link sharing, priority and traffic shaping.

For this purpose, an algorithm for hierarchical link-sharing and traffic shaping based on the Token Bucket (TB) model has been proposed in the paper. This algorithm is much simpler than other hierarchical link-sharing algorithms such as CBQ. But it fulfils the link-sharing tasks similar to these done by CBQ and adds the feature of hierarchical traffic shaping. The novel algorithm presented in this paper is based on the concept of hierarchical link-sharing and each class in the hierarchy is bound with a token bucket, which can shape the traffic of

this class, limiting the average rate and burst volume. However, some classes do not need traffic shaping. This token bucket based scheduling algorithm was then improved to enable these classes to borrow excess bandwidth, so as to make better use of the total bandwidth. The analysis and experiment results show that through the way of weighted fair token sharing, this algorithm guarantees the basic bandwidth service for each class and enables weighted fair sharing of excess bandwidth. Therefore, it performs efficient hierarchical link-sharing. The algorithm complexity is $O(N)$, where N is the number of the total leaf classes. For home routers or gateways, the number of classes in the algorithm is relatively small, and thus easy to implement.

REFERENCES

[1] Wan-Ki Park, Sung-Il Nam, Chang-Sic Choi, Youn-Kwae Jeong, and Kwang-Roh Park, "An implementation of FTTH based home gateway supporting various services", IEEE Transactions on Consumer Electronics, Vol. 52, No. 1, pp. 110 – 115, February 2006.

[2] Kyoung-Youn Cho, and Kwang-ho Choi, "A novel architecture of Home gateway for efficient packet process", Proceedings of IEEE Workshop on Knowledge Media Networking, pp. 63 – 67, July 10-12, 2002, Kyoto, Japan.

[3] Satish Gupta, "Home gateway," White Paper of Wipro Technologies, August 2004. Available: http://www.broadcastpapers.com/whitepapers/wiprohomegateway.pdf.

[4] F. T. H. den Hartog, M. Balm, C. M. de Jong, and J. J. B. Kwaaitaal, "Convergence of Residential Gateway Technology", IEEE Communications Magazine, Vol.42, No.5, pp138-143 , May 2004.

[5] Wen-Shyang Hwang and Pei-Chen Tseng, "A QoS-aware Residential Gateway with Bandwidth Management" , IEEE Transactions on Consumer Electronics, Vol. 51, No. 3, pp. 840-848, Aug. 2005.

[6] Wan-Ki Park, Chang-sic Choi, II-woo Lee and Jonghyun Jang, "Energy efficient multi-function home gateway in always-on home environment", IEEE Transactions on Consumer Electronics, Vol. 56, No. 1, pp.106-111, February 2010.

[7] Y. Royon, and S. Frénot, "Multiservice home gateways: business model, execution environment, management infrastructure", IEEE Communications Magazine, Vol. 45, No.10, pp. 122-128, October 2007.

[8] B. Hardjono, A. Wibisono, A. Nurhadiyatna, I. Sina and W. Jatmiko,"Virtual detection zone in smart phone, with CCTV, and Twitter as part of an integrated ITS, " International Journal on Smart Sensing and Intelligent Systems,Vol. 6, No. 5 pp. 1830 – 1868, Dec.2013.

[9] Seppo Hätönen, Aki Nyrhinen , Lars Eggert, Stephen Strowes, Pasi Sarolahti and Markku Kojo, "An experimental study of home gateway characteristics", Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, pp.260-266, November 2010, Melbourne, Australia.

[10] San-Pil Moon, Joo-Won Kim, Kuk-Ho Bae, Jae-Cheon Lee, and Dae-Wha Seo, "Embedded Linux implementation on a commercial digital TV system", IEEE Transactions on Consumer Electronics, Vol. 49, No. 4, pp. 1402-1407, November 2003.

[11] Seongsoo Hong, "Embedded linux outlook in the PostPC industry", Proceedings of sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, pp. 37-40, May 14-16, 2003, Hokkaido, Japan.

[12] S. Floyd, and V. Jacobson, "Link-sharing and resource management models for packet networks", IEEE/ACM Transactions on Networking, Vol. 3, No. 4 , pp. 365-386, August 1995.

[13] M. Bechler, H. Ritter, G. Schafer, and J. Schiller, "Traffic shaping in end systems attached to QoS-supporting networks", Proceedings of IEEE Symposium on Comput. and Commun. , pp. 296-301, 2001.

[14] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang and W. Weiss "An Architecture for Differentiated Services", RFC2475, The Internet Society, December 1998, Available: http://www.hjp.at/doc/rfc/rfc2475.html.

[15] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks", ACM SIGCOMM Computer Communication Review , Vol. 28, No. 4, pp.118-130 ,October 1998.

[16] P. F. Chimento, "Standard Token Bucket Terminology", May 2000, Available: http://qbone.internet2.edu/bb/Traffic.pdf .