

# Estimate and Serve: Scheduling Soft Real-Time Packets for Delay Sensitive Media Applications on the Internet

Ishan Vaishnavi, Dick C.A. Bulterman  
Centrum voor Wiskunde en Informatica, Science Park 123, 1098 XG, Amsterdam  
{i.vaishnavi,dick.bulterman}@cwi.nl

## ABSTRACT

This paper presents a new scheduling algorithm for real time network delivery of packets over Diffserv networks for delay sensitive applications. We name the networks that support this algorithm as Estimated Service (Estserv) networks. These networks, for real time packets, estimate the probability of the packet meeting its deadline and schedule it according to this estimation. This paper validates, given this estimation mechanism, the better performance of the scheduling algorithm over traditional solutions. We show that using Estserv for delay sensitive applications, we can provide out-of-band scheduling, save bandwidth on packets with expired deadlines and handle bursts without losing the scalability of Diffserv. We show with the help of an implementation in the Linux kernel's ip-forwarding part, that, given the estimation value, Estserv performs better than Diffserv in terms of deadlines, while still saving bandwidth.

## Categories and Subject Descriptors

C.2.1 [Computer Communication Networks]: Network Architecture and design; C.2.4 [Computer Communication Networks]: Distributed Systems, distributed applications

## General Terms

Algorithms, Design, Experimentation

## Keywords

Real Time Networks, Distributed Systems

## 1. INTRODUCTION

Today, a number of applications require soft real-time delivery of packets over a network. Such applications include: i) *Tele-immersive environments* [7], ii) *VoIP*, iii) *Distributed Gaming*, iv) *Distributed Shared Experiences*. Non-interactive applications, such as streaming, are delay tolerant and focus

on reducing jitter. The above mentioned applications are inherently interactive in nature and thus delay intolerant. Experience has shown us that data expands to the size of the network and the smartest of bandwidth allocations are often inadequate, even more so with the new multimedia applications such as tele-immersion. This means that packets belonging to these applications may frequently require control-flow-like out of band transmissions. This is difficult using the traditional QOS architectures, Intserv and Diffserv, which mainly focus on keeping the jitter constant and arrival rates predictable. Furthermore, while some applications, like gaming, can submit constant sized packets at a periodic rate, other applications may deliver unpredictable sizes of data, sometimes instantaneously. While Intserv will only allow such cases if the maximum has been reserved for, Diffserv cannot handle bursts due to short queue length. In this paper we look at the drawbacks in Diffserv and how Estserv improves on them while preserving scalability.

As we shall see the design of Estserv is divided in two parts: i) the estimation mechanism ii) the scheduling mechanism. This paper presents the latter as a qdisc implementation in Linux and the results thereof.

## 2. MOTIVATION

How are delay sensitive applications programmed today ? During the session setup the system pings all the participating nodes and then calculates the worst case round-trip time. Based on the resulting time the system informs the user if it is suitable to continue with the application. If a participating node losses its connection during the session, the entire session is paused across all nodes. While this is a simple and scalable approach it is reactive and not proactive in nature. The approach does not describe how these packets are scheduled at individual nodes for forwarding. As we shall see this approach forms the first step in Estserv networks.

Traditional solutions to provide proactive QOS/Real Time (RT) support over networks are Intserv[4] and Diffserv[2]. While Intserv aims to achieve hard RT characteristics over the Internet by restricting excess traffic and policing the accepted traffic, Diffserv only provides a best effort service with priority scheduling for higher classes of service, and, policing on the edge routers. Intserv has not been widely accepted in implementation due to scalability issues. Providers use Diffserv based networks to police and shape traffic incoming to their networks and may provide limited RT support with the Expedited Forwarding (EF)[5] class, subject to the contract. While the EF class gives user's a relatively

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

NOSSDAV09, June 3–5, 2009, Williamsburg, Virginia, USA.  
Copyright 2009 ACM 978-1-60558-433-1/09/06 ...\$5.00.

free highway through their network, it can get congested on the edges and at inadequately provisioned routers with high fan-in. In such cases the shortcomings of the Diffserv architecture for RT support become apparent:

1. *Out of Band Forwarding* Packets are scheduled within the EF class as FIFO and suffer, albeit short, but avoidable queueing delays due to other equal priority packets. Let us explain this with an example. Assume a node has two packets  $p_1$  and  $p_2$  in its queue. Assume that  $p_1$  arrived just before  $p_2$ . Then it will process the packet  $p_1$  first. This may not be optimal, since the path further along  $p_2$ 's connection may be busier than that taken by  $p_1$ . Thus while  $p_1$  may reach its destination too early,  $p_2$  may be too late.
2. *Wasted Bandwidth* A packet will still be transmitted through the network even if it is discarded at the destination due to expired deadline. This is *wasted bandwidth* and can be used at congested nodes for transmission of more successful packets.
3. *(Un)Predictability in during bursts* Due to short suggested queue lengths in the EF class[5], Diffserv does not perform well with bursts which may overrun the allotted send rate even if the connection sticks to the flow specification at an average over large periods.

This paper presents a mechanism (Estserv), which our results show performs better than Diffserv w.r.t. these three points.

A good explanation of the Diffserv EF and the effect of various network implementation factors on delay and jitter is presented in [6]. Papers [9, 8] study various scheduling algorithms at a router to schedule packets between various Diffserv classes or per hop behaviours (PHBs), namely, EF, Assisted Forwarding (minimum 3 sub classes) and best effort (BE), while being fast enough for EF, reliable for AF and increasing BE throughput. We look at the scheduling of packets within the EF class and are not concerned with how inter class scheduling is done. For our experiments we use the Hierarchical token bucket filter explained at<sup>1</sup>. Most research has not looked at scheduling algorithms within the EF class since it would introduce further processing delays in the RT channel. However, we contest this claim on the basis that RT doesn't necessarily mean fast, and if we can look at the timing constraints to be met by the packets being transmitted, we can perform efficient scheduling which outweighs the processing delay as we show in this paper.

### 3. ESTIMATE AND SERVE

#### 3.1 Theoretical Background

Network Calculus [3] looks at a router as a reference point and analyses the different streams of packets passing thorough it with various arrival and departure rates. We take a different approach. We choose the packets as a reference point and analyse the time it needs to pass through various nodes. This gives us an estimate (probability) of whether the node meets its deadline. Thus, if, packets carry their deadlines with them a comparison can be made and the packet accepted or rejected based on this estimate.

Let us imagine a Task  $T$  with minimum deadline time from creation  $d$ . Let us say the task is created at node  $n_A$

<sup>1</sup><http://luxik.cdi.cz/devik/qos/htb/>

to be executed at node  $n_B$ . Assume that there are  $N+1$  hops from node  $n_A$  to  $n_B$  (thus  $N$  nodes in between). Each node takes  $t_{fwd}$  time to forward a packet.  $P_A$  and  $P_B$  are processing times at  $n_A$  and  $n_B$ . Except  $d$ , all variables can be considered as random. Thus, if

$$d > N\overline{t_{fwd}} + \overline{P_A} + \overline{P_B} + (N+1)C \text{ where } \overline{y} = \max(y) \quad (1)$$

where  $C$  is some transmission constant per link, then we can give a hard-real-time guarantee that the process will be executed. This implies

$$N < \frac{(d - C - \overline{P_A} - \overline{P_B})}{(\overline{t_{fwd}} + C)}. \quad (2)$$

This means that if the deadline characteristics and the other aforesaid upper bounds are known one can design a distributed network with hard RT processing subject to the limitation that the distance between two nodes in the network which will take part in RT communication is no more than  $N+1$ , where  $N$  is the highest integer value satisfying 2. In reality, however, these upper bounds are not guaranteed, therefore a hard RT system cannot be designed. However, can we compute the probability of a deadline miss? If so, can we design a distributed soft RT system with a known probability of deadline miss.

The probability density function for the time taken to communicate and process a network packet is a multi-variate  $N+2$  dimensional function given by

$$P(\text{time} - \text{taken}(t)) = f(t_{fwdl}, P_A, P_B) \quad \text{where } l = 1 \dots N \quad (3)$$

At a macroscopic level we want to know the probability of the total time taken being less than the deadline,  $d$ . Since all the times are added to yield the final time and each of these time variables is independent<sup>2</sup> the probability density function is given by

$$p(t = x) = \int_0^x p_1(t_{fwd} = t_1) \dots \int_0^x p_i(t_{fwd} = t_i) \int_0^x p_N(t_{fwd} = t_N) \int_0^x p(P_A = t_A) \dots p(P_B = x - (\sum_{i=1}^N t_i + t_A)) dt_1 \dots dt_N dt_A \quad (4)$$

Thus the cumulative probability is then simply

$$p(t < d_i) = \int_0^{d_i} p(t = x) dx \quad (5)$$

Thus if these distributions are provided then the system can estimate the reliability for a given value of deadline or conversely minimum deadline required to give the user a certain system reliability. We term this probability distribution for a node as *Node Network Forwarding Character* (NNFC).

Considering the operations at a micro view, what is the scheduling mechanism a node must employ for delivering these packets, given each packets global deadline,  $d$ , the packet arrival time  $t_{arr}$  and and the fore mentioned NNFC of all the nodes in the connection. A probability, of the packet meeting its deadline, can be computed by the same equations as 4 and 5 above, by substituting global deadline  $d_{new}$  as  $d_{new} = d - t_{arr}$  and replacing  $P_A$  with the forwarding

<sup>2</sup>The time taken by a router to transfer a packet is independent of the time taken by other routers.

time for the current node. Thus the cumulative probability  $p(t < d_{new} \{= d - t_{arr}\})$  can be calculated. The lower this probability, of a packet meeting its deadline, the higher the incoming message's priority, this is the scheduling mechanism. However, if this priority is too low then the packet is dropped altogether.

### 3.2 Algorithm

The approach described in the theory is not scalable, since it requires a node to know all the other NNFCs. To solve this we divide the approach in two parts:

#### Estimation Part

- During *connection setup phase* a client “estimates” the total delay to its destination node and the number of hops. Based on this, the *client application* accepts or rejects the connection. The actual deadline,  $t_{dl}$  and the connection's required reliability  $R$  form the part of the packet's IP-header options if it is a RT channel during the communication phase. The (number of hops + 1) is the value placed in place of the ip TTL field.
- During the *communication phase*, each router maintains a minimum acceptable per-hop estimate,  $p_{est}$  required to transmit the packet<sup>3</sup> and adds to it its NNFC (it's own forwarding time estimate,  $t_{for}$ ). Thus it may reject a packet if

$$\frac{(t_{dl} - t_{now})}{TTL} \leq f(p_{est}, t_{for}, R) \quad (6)$$

#### Scheduling Part

- Network control packets including Time synchronisation packets have highest priority.
- RT packets are next. Within the RT packets queue packets are scheduled as a partial order  $O = \{p_1, \dots, p_i, \dots, p_j, \dots, p_n\}$  over packets,  $P = \{1..i, \dots, j, \dots, n\}$  such that

$$p_i \leq p_j \Leftrightarrow \frac{(t_{dl_i} - t_{now})}{TTL_i} \leq \frac{(t_{dl_j} - t_{now})}{TTL_j} \quad \forall i, j \in P \quad (7)$$

- For routers running RTOS the deadline,  $d$ , before which the next packet must be dequeued is simply

$$d = f(p_{est}, t_{for}, R) - \frac{(t_{dl} - t_{now})}{TTL} \quad (8)$$

Eqn.s 6 and 7 implicitly imply that the network needs to be time synchronised<sup>4</sup>. The focus of this work is on the scheduling part, leaving the estimation part outside of the scope of this particular contribution. It is important to notice that the estimate  $f(p_{est}, t_{for}, R)$  need not be calculated by the network, but can be an administrator set limit.

## 4. IMPLEMENTATION AND SETUP

### 4.1 Linux Implementation Details

The core idea of the implementation is that each packet's header is marked with its associated deadline and its TTL field is set to the expected number of hops. When the packet

<sup>3</sup>This may or may not be separate for separate entries in the routing table and can be calculated dynamically using the same estimation mechanism in step1 before.

<sup>4</sup>Alternatively, every node subtracts the time it took in processing the packet. This is however considered slow.

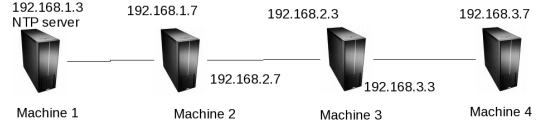


Figure 1: Experimental Setup

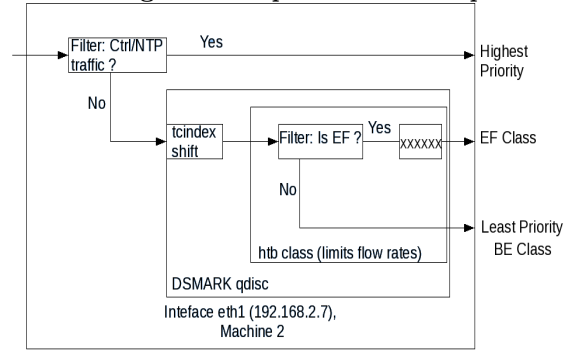


Figure 2: Setup A) XXXXXX = FIFO. Setup B) XXXXXX = Estserv

is received by a router, it checks how much time the packet has left per hop to meet its deadline. If this value is below a certain threshold, that we call ESTIMATE ( $f$  in eqn. 6), then this packet is dropped. Otherwise it is inserted in the appropriate place in the delivery queue, which is sorted based on this time. We implemented our scheduling algorithm as a module in the Linux kernel. It is accessed via the  $tc$ (traffic controller) utility. A scheduling algorithm implementation is called a *queuing discipline* in the Linux Kernel. Multiple queuing disciplines are already implemented in the Kernel, such as FIFO, priority, class based queuing and more. For more detailed explanation of these concepts please refer [1]. All Linux qdiscs implement a number of functionalities. The important ones for our explanation are the enqueue and the dequeue function. A detailed explanation of the rest is found in [1]. Thus, a FIFO qdisc would enqueue at the tail, dequeue from the head and drop when the queue is full.

#### The Estserv Qdisc

The Estserv queue implements the algorithm we described in Section 3.2 Eqn 7. We currently use a linear insertion sort over the appropriate  $sk\_buff$ (Kernel packet structure) queue in the enqueue function, a later implementation will be a min heap structure over the queue. The dequeue function returns the next packet in the queue whose deadline has not expired while waiting in the queue.

### 4.2 Experimental Setup

We created a linear topology network of 4 machines as shown in Fig 1. Appropriate routes are added to the routing table using the “route” Linux utility. Machines 2 and 3 are running modified and normal Centos(2.6.10 kernel) and machine 1 and 4 are laptops running Fedora 7. In this topology we study machine 2's behaviour as a Diffserv router A)without Estserv B) with Estserv. In the sections that follow we refer to *Diffserv without Estserv as Diffserv* and *Diffserv with Estserv as Estserv* for brevity. At first all nodes are time synchronised using NTP. Since the experiment runs only for 35 seconds each time, it is not useful to run NTP

period p (in ms)	No. of clients	Bandwidth(BW) (Mbps)
35	29	0.733
50	40	0.71
75	60	0.71
100	85	0.75
150	125	0.74
200	170	0.75
300	245	0.723

Table 1: Bandwidth duplicated for machine3, 4.

daemons, an initial sync. is enough. This was achieved using the “ntupdate” utility. Manual comparison showed that the clocks were within 1 ms of each other the experiments.

Using tc on the machine 2 interface eth1 we first set up a Diffserv router, then an Estserv router. Fig 2 shows this setup. The box marked XXXXXX in the figure is a qdisc, it is by default FIFO for Diffserv. In our setup we set it to our Estserv qdisc. The figure shows a prio (priority) qdisc for ctrl/NTP traffic followed by a DSMARK (Diffserv) qdisc with a *tcindex*<sup>5</sup> filter followed by an htb qdisc of maximum bandwidth ceiling 10Mbps. 10Mbps is was chosen for this experiment so that the results are not seriously affected by scheduling and other delays on the traffic generating machine<sup>6</sup>. The htb qdisc consists of 2 classes the EF class and the Best Effort class. The EF class is given a reserved bandwidth of 5Mbps while the BE class is given a bandwidth of 5Mbps. At first we send BE traffic using the nuttcp utility from node 1 to node 3. nuttcp tries to send UDP packets to machine 3 and measure the approximate throughput.

We then start RT traffic from 1 to 4 and from 1 to 3. A single RT client application is a simple c++ application that sends 58 byte packets every period (= deadline) with the deadline in the data field of ip and the TTL manually set as number of hops per destination. The clients are duplicated for destination machine 3 and machine 4, with different hop counts of course. For our experiment we used the setting in Table 1. From the 7 clients shown 6 are started simultaneously at (t=0), these form stable traffic. One of the 7 clients forms the bursty traffic and is started at (t=5secs) and killed after 5 seconds this is repeated for (t=15) and (t=25). The experiment is run for a total of 35seconds. The varying traffic makes the send rate vary across the allocated 5 Mbps bandwidth. We shall present results of the bursty traffic having deadline i) short(0.035ms) ii) intermediate (0.1ms) and iii) long (0.3ms). Notice that the minimum deadline of 35ms is for hi-quality audio/video conferencing and that of 300 ms is for virtual world applications and that the bandwidths are somewhat equally divided across periods. Different deadlines yielded similar results for the same value of total bandwidth. The different values of the period can also be seen as packets arriving from far away sources.

Execution 1 shows that when all the clients are executed, real time traffic expands to take the place of the nuttcp traffic to a maximum of 5Mbps. The behaviour of the network so far was controlled by the htb and the DSMARK qdiscs, which are common to both scripts thus we don’t see a difference in them so far. Going further, we look only at the EF (RT) part of the traffic and ignore the nuttcp generated traffic for now.

<sup>5</sup> *tcindex* is a Diffserv Code Point (DSCP) reading filter

<sup>6</sup> Generating 1Gbps causes unpredictable delays on machine1

## Command Execution 1 RT Traffic expands to its BW

```
[root@ishan ~]# nuttcp -R15M -T10m -i1 -uv4 192.168.2.3
1.1328 Mb / 1.00 sec = 9.4950 Mbps ...<truncated>
0.7891 Mb / 1.00 sec = 6.6138 Mbps
0.5889 Mb / 1.00 sec = 4.9358 Mbps
```

## 5. RESULTS

At first we need a way of measuring which RT channel is better. Throughput is not a good measure for RT transport, simply because it doesn’t express any temporal information. Instead we measure the ratio of the *deadlines met* per connection and the *wasted bandwidth*. *Wasted bandwidth* is defined as the bandwidth used by the packets at machine 2 which are rejected by the destination.

### 5.1 Ideal Network: No Delay No Jitter

The table 2 shows the wasted bandwidth and the total number of met deadlines on node 4. Due to space constraints we only present results machine 4. Wasted bandwidth is the average amount of bandwidth over the 35 second interval wasted at node 2 in transporting packets which are rejected by the destination node. Packets are rejected at the destination node if the delay encountered over the network is higher than 105% of the period i.e. we take 5% of the deadline as acceptable jitter. The calculations are made from the point of view of machine 4 subject to the synchronisation it has, i.e., the results are not normalized for different synchronization levels across experiments.

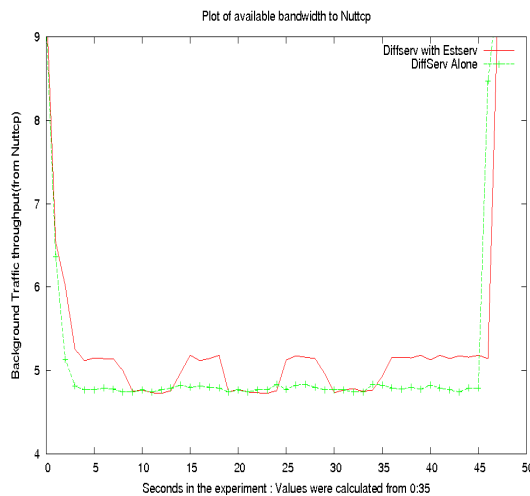


Figure 3: Bandwidth Saved by Estserv

Table 2a) shows that Estserv (qlen = 500kb) outperforms Diffserv with a large queue (50kb) in EF class and performs better than EF with a short queue length in terms of deadlines met with almost the same level of wasted bandwidth. This experiment clearly shows and verifies the reason for [2] to suggest small queues for Diffserv EF class and is thus only presented here for 2 a) case. The results in Table 2 show some interesting issues. In case a) Estserv in general seems to outperform Diffserv. Estserv even performs slightly better in the 35ms case which is the burst traffic without seriously affecting any other classes. This is somewhat an unexpected result, since we predicted Diffserv to be bogged down by the transmission of the 35 ms packets. But while

qdisc		Period (in ms)						
		35	50	75	100	150	200	300
a)Bursty traffic deadline = 0.035s								
A) $qlen = 50k$	Avg., Std. Dev.(in ms)	85, 7	68, 33	68, 33	68, 33	68, 32	69, 32	70, 31
	Wasted BW(bps)	340885	275801	267621	0	0	0	0
	%age Deadlines met	0.9	19	19	92	91.7	91	89.7
A) $qlen = 10pkts$	Avg., Std. Dev.(in ms)	0.4, 0.7	0.6, 3.7	0.6, 3.7	0.6, 3.6	0.6, 3.7	0.6, 3.8	0.6, 3.5
	Wasted BW(bps)	0	371	0	0	0	0	0
	%age Deadlines met	53.7	93.4	95	94	93.6	92.5	90.6
B) $f = 0, q = 500k$	Avg., Std dev (in ms)	11.2, 9	5, 6	5, 6.2	6.1, 7.7	32.2, 23	73, 38	159, 68.1
	Wasted BW(in bps)	0	0	0	0	0	0	0
	%age of Deadlines met	58.3	97.5	96.9	96.8	96.4	95	92.9
b)Bursty traffic deadline = 0.1s								
A) $qlen = 10pkts$	Avg., Std. Dev.(in ms)	0.68, 4	0.6, 3.8	0.7, 3.8	0.4, 0.8	0.7, 4.1	0.6, 3.8	0.7, 4.1
	Wasted BW(bps)	1816	556	0	0	0	0	0
	%age Deadlines met	94	93.6	94.3	57.6	93	92	90.6
B) $f = 0, q = 500k$	Avg., Std dev (in ms)	19.9,19.7	26.1, 18.9	26.4,18.2	36.5, 8.18	30, 20.2	68, 36	151.5, 71
	Wasted BW(in bps)	571	1206	755	0	821	715	715
	%age of Deadlines met	67	96.8	96.9	100	95.9	94.7	92.8
c)Bursty traffic deadline = 0.3s								
A) $qlen = 10pkts$	Avg., Std. Dev.(in ms)	0.69, 3.8	0.76, 3.8	0.68, 3.8	0.67, 3.6	0.68, 3.7	0.69, 3.7	0.44, 1.2
	Wasted BW(bps)	1749	517	0	0	0	0	0
	%age Deadlines met	92	88.4	92	91	90.6	89	89.6
B) $f = 0, q = 500k$	Avg., Std dev (in ms)	20.6, 13	26.6, 13	26.8, 13	30.9, 14	53.3,28.6	99.4, 37	195.5, 8
	Wasted BW(in bps)	2301	2280	238	0	0	0	0
	%age of Deadlines met	68.3	97.5	95.8	96.4	95.4	94.1	99.9

**Table 2: Avg., Std. Dev. of Delay , Wasted bandwidth and Deadline results for A.) Diffserv Alone B.) Diffserv with Estserv( $f = 0$ ). A value of 0 is used as no delay is simulated.**

qdisc		Period (in ms)						
		35	50	75	100	150	200	300
a)Bursty traffic deadline = 0.035s								
A) $qlen = 10pkts$	Avg., Std. Dev.(in ms)	59, 4.4	59, 5.8	59.1, 5.4	58.9, 4.5	58.7, 4.4	58.9, 4.4	58.9, 4.6
	Wasted BW(bps)	301909	280826	928	0	0	0	0
	%age Deadlines met	0	7.4	84.9	89	90.7	90	89.5
B) $f = 16.6$	Avg., Std dev (in ms)	NoTx	51.4, 0.9	58.9, 4.5	58.8, 4.5	58.6, 4.5	58.8, 4.3	58.8, 4.5
	Wasted BW(in bps)	0	888	0	0	0	0	0
	%age of Deadlines met	NoTx	2.3	90	92.5	94	93.8	93
b)Bursty traffic deadline = 0.1s								
A) $qlen = 10pkts$	Avg., Std. Dev.(in ms)	58.9, 6.	59, 6	59.3,7	59, 4.6	58.8, 5	58.8, 4.3	58.8, 4.5
	Wasted BW(bps)	297994	275814	662	0	0	0	0
	%age Deadlines met	0	7.5	76	91.5	89	89.4	88.7
B) $f = 16.6$	Avg., Std dev (in ms)	NoTx	51, 0.9	59, 4.5	59.2, 4	59.3, 4.6	58, 4.4	58.8, 4.5
	Wasted BW(in bps)	0	371	0	0	0	0	0
	%age of Deadlines met	NoTx	1.4	91	99.3	94.5	94.4	93.7
c)Bursty traffic deadline = 0.3s								
A) $qlen = 10pkts$	Avg., Std. Dev.(in ms)	59, 5.6	59, 5.8	59, 5.2	59, 4.5	59, 4.5	59, 4.5	59.2, 12
	Wasted BW(bps)	301348	281117	477	0	0	0	154
	%age Deadlines met	0	6.9	84.7	88	89.8	89	89
B) $f = 16.6$	Avg., Std dev (in ms)	NoTx	51.4, 2	59, 4	58, 4.5	58.7, 4.5	58.8, 4	58, 5.5
	Wasted BW(in bps)	0	636	185	0	0	0	0
	%age of Deadlines met	NoTx	1.5	90	92	93.5	93.3	99.9

**Table 3: Results for incoming jitter and forwarding delay at machine 3. A.)Diffserv alone( $qlen = 10pkts$ ) B.) Estserv( $f = 16.6ms$ ), ( $qlen = 500kb$ ). NoTx = No Transmissions**

it's performance in the other cases is somewhat worse it also under performs in the 35ms case. We believe this is because of mac RTS/CTS collisions owing to a sudden burst of very short deadline traffic. But a more thorough investigation is required, however, that is not the focus of this paper.

In case b) and c) Estserv under performs in the 35ms deadline case and performs much better for the other deadlines. Contrary to intuition, this is a desired and expected behaviour for a RT channel. When the network is not too busy with just minor queuing the 35ms packets should be transmitted first. However, when the network is busy, a RT channel must not sacrifice the performance of all the connections in it. Instead it must throttle one while keeping the others stable, so that they can still continue. The choice of which channel to throttle is in this case decided by the most demanding channel, i.e., the one with the lowest deadline. On the other hand Diffserv (A) distributes the drop in

performance to all channels, which may result in all of the associated applications performing unacceptably.

The table also shows out-of-band queuing for Estserv. Look at the avg delays across deadlines. While Diffserv alone has a constant delay for all channels, in Estserv's case the delay increases with the deadline value. This is almost Intserv like behaviour, except that it is work conserving. This way all packets must wait their appropriate turn to be transmitted and a short deadline connection cannot block the long deadline ones indefinitely by hogging the channel.

#### Command Execution 2 RT Traffic expands to its BW

```
[root@ishan ~]# tc qdisc add dev eth1 root netem \
    delay 50ms 1ms distribution normal
```

We also notice some small values of wasted bandwidth for the Diffserv case, which shouldn't be possible in a no delay no jitter network. This is due to scheduling issues on

one of the Linux machines, since they are not RT operating systems. Similarly the same values for Estserv are slightly higher than Diffserv. In addition to the scheduling errors these are errors due to a slight underestimation of the ESTIMATE value. It is not manually possible to choose the most efficient ESTIMATE value. Future work on the estimation part in Section 3.2 will address this issue. Manual verification revealed that most of these values were borderline cases. Lastly, as mentioned in Section 2, Diffserv doesn't handle short bursts of bandwidth very well. It considerably underperforms w.r.t. Estserv in the burst period channel in all the cases. The results above were for an *ideal network* with no jitter or delay. Even here we already saw Estserv perform better overall from a RT perspective.

## 5.2 With Jitter and Delay

To simulate jitter on the incoming interface (from machine 1) we add a random sleep time  $\leq 15\text{ms}$  to the clients before they send out the packets. Furthermore, on the path from machine2 to machine 4, machine3 now suffers a delay of 50ms with a 1ms normal delay distribution. This is done using the network emulator qdisc, Command 2. For Estserv this represents a ( $f = (50/3) = 16.6\text{ms}$ ) delay per hop from machine 1 to 4. In this case while Diffserv would still continue transmitting the 35 ms packets Estserv will reject the 35 ms connection attempts, saving bandwidth. Table 3 shows the results in this case for all three burst profiles.

We highlight first the bandwidth saved by Estserv. Since Estserv rejects the 35ms class altogether it saves considerable amount of bandwidth. In 50ms deadline case as well Estserv saved a considerable amounts of bandwidth. Where did all this bandwidth go? Recall from Section 4.2 that we are also running background nttcp traffic at 10Mbps. Figure 3 shows the plot of the background traffic throughput. The graph shows that there is more bandwidth available at the RT channel in the Estserv case, which is taken over by nttcp traffic. Thus the channel can now have additional bandwidth at it's disposal whereas FIFO is full. This validates point 2 of the shortcomings listed in Section 2.

The fact that Estserv performs under bandwidth can also be noticed in the average delay and std. dev. values in the table. Notice that different from the previous case there is almost no variation here. In fact the values seem to be identical to the Diffserv case. This is simply because, in this case, there is not much of queuing experienced by Estserv. This adds up since it is transmitting at below link capacity after rejecting the 35ms connection. A manual verification showed us that the queue length in all the 3 cases here never crosses 50 packets. This implies that some of the bandwidth has gone in improving the deadline performances of the other connections as is clear from the table, since there is very little queueing. Except for the 50ms case where owing to the Estimate value Diffserv drops most packets, it performs significantly better than Diffserv in other deadline classes. We also see that Diffserv is again significantly outperformed in the Burst class in Table 3b) and c) showing its intolerance to short bursts.

Overall the results showed that Estserv provides out of band queuing for Diffserv networks based on the packets ability to meet its deadline, without losing scalability. This, in turn, brings all the advantages of having a queue, such as, better capability to handle bursts and in some cases better performance. We also demonstrated that by dropping pack-

ets which were unlikely to meet their deadlines, provided the estimation mechanism, Estserv can save bandwidth.

## 6. ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 216647. The project's website is at [www.inem4u.eu](http://www.inem4u.eu).

## 7. CONCLUSIONS AND FUTURE WORK

Taking a step away, from a more broader perspective with Estimated Service we are trying to introduce a new approach for scheduling soft RT tasks, for which the probability (frequency) distribution of the time taken is known. These tasks can be scheduled at much lower time than the worst case requirement. We call this technique "probabilistic scheduling". This can also be applied to other scheduling systems.

We presented three drawbacks of the traditional Diffserv EF PHB for RT applications, namely i) no out of order scheduling ii) inability to predict deadline expiration iii) inability to handle bursty traffic. In this paper we proposed a new scheduling algorithm for use within the EF PHB of the Diffserv class. We validated with the help of a real Linux implementation that, provided, there is a good estimation mechanism, extending Diffserv with our scheduling algorithm performs better for packet delivery in delay sensitive multimedia applications, than Diffserv alone. Future work, after these encouraging results, is to concentrate on the design and implementation of an effective estimation algorithm. We are also in the process of looking for good data sets which provide real data with real deadlines or alternatively, experimental deployment platforms.

## 8. REFERENCES

- [1] W. Almesberger, *Linux network traffic control - implementation overview*, 1999.
- [2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, *Rfc 3260 - An architecture for differentiated services*.
- [3] Jean-Yves Le Boudec and Patrick Thiran, *Network calculus: A theory of deterministic queuing systems for the internet*, Springer Verlag - LNCS 2050, 2004.
- [4] R. Braden, D. Clark, and S. Shenker, *Rfc 1633 - Integrated services in the internet architecture: an overview*.
- [5] B. Davie, A. Charny, J.C.R. Bennett, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis, *Rfc 3246 - An expedited forwarding phb*.
- [6] T. Ferrari and P.F. Chimento, *A measurement-based analysis of expedited forwarding phb mechanisms*, Proceedings of IWQoS, 2000.
- [7] K. Nahrstedt, R. Diankov, R. Bajscy, Z. Yang, B. Yu, and W. Wu, *A study of collaborative dancing in tele-immersive environments*, IEEE Symposium on Multimedia (2006).
- [8] A. Ghaffar Pour Rahbar and O. Yang, *Lgrr: A new packet scheduling algorithm for differentiated services packet-switched networks*, Elsevier Computer Communications Journal, 2008.
- [9] Mei Yang, Jianping Wang, Enyu Liu, and S. Q. Zheng, *Hierarchical scheduling for the diffserv class*, IEEE Globecom, 2004.