INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. • Suite 600 • Berkeley, California 94704-1198 • (510) 643-9153 • FAX (510) 643-7684

Introducing resources management in IP-based nodes

Pietro Manzoni*

TR-94-040

October 1994

Abstract

The Internet Protocol was designed to be used with packet-switched communication networks and, as originally designed, does not provide the characteristics necessary to support voice and video transmission. The lack of control over the amount of connections supported leads to highly variable delays for packets and often to packet loss.

In this paper, an enhancement of an IP based node (called IP') is presented to allow a simple management of the node's resources. We introduce higher interaction between the transport and the network layers through additional processes and functions. The paper also presents, as an example, a transport layer protocol that shows how to take advantage of the new functionalities provided by the IP' nodes.

Two fundamental hypothesis throughout the design process were: 1) the effort in moving an IP-based node to an IP'-based node had to be smaller than the effort required in moving to a completely different protocol suite, and 2) the regular Internet traffic should not be affected or modified at all.

Simulations results are presented to show that this approach can actually bound the variation of delay and throughput. In addition this approach can also control the number of packets lost.

*Politecnico di Milano, P.za L. Da Vinci, 32 - Milan, Italy 20133. manzoni@elet.polimi.it

1 Introduction

New multimedia applications like conferencing tools or image databases used for scientific visualization have been developed that need to exchange voice and video data.
A real-time communication (RTC) service can be defined as a computer communication in which clients (application programs) can specify performance requirements and obtain guarantees about their fulfillment. By performance requirements we refer to *throughput* and *delay*. We want the network to be capable of limiting the variability in the provided throughput and also to limit the range of values among which the packets' delay can vary. These two characteristics are referred to as *deterministic throughput bound* and *deterministic delay bound*[1].
Internet, the most widely used internetwork, principally uses IP as a network layer protocol. IP was designed for use in interconnected systems of packet-switched communication networks and, as originally designed, did not provide the throughput and delay characteristics necessary to support voice and video applications.
In packet-switched networks, routers (or intermediate communication nodes) are shared among many different "conversations" at the same time. IP however, does not require routers to maintain state information describing the streams of packets flowing through them. By *conversation* we mean what a connection-oriented protocol would call a *connection*. The lack of control over the amount of *conversations* supported, leads to highly variable delays for packets and, often to packet loss. Techniques that have been adopted to limit this *congestion* situation [2, 3] do not completely solve the problem.
In this paper an enhancement of an IP node is presented to allow a simple management of the node's resources. A higher interaction between the transport and the network layers, provided by additional processes and functions, support and coordinate their work and the managing of the node's resources. The new IP node will be referred to as IP'.
Two fundamental hypothesis throughout the design process were: 1) the effort in moving an IP-based node to an IP'-based node had to be smaller than the effort required in moving to a completely different protocol suite, and 2) the regular Internet traffic should not be affected or modified at all.
This paper also presents, as an example, a transport layer protocol that shows how to take advantage of the new functionalities offered by the IP' nodes. This protocol is a connection-oriented real-time version of UDP called *RDP*. *RDP* is a light-weight transport protocol that uses a rate-based flow control and allows the user to obtain higher performance from the network.
This work starts from an evaluation of the following other approaches: *SRP* [4], *RTP* [5], *ST-II* [6], *FLOWS* [7], *RSVP* [8] and *Tenet* Suite-1 [9, 10] (a detailed comparisons can be found in [11]).
The paper is organized as follows: in Sect. 2 the characterization of data streams is presented. Section 3 shows the structure of the IP' based nodes. Section 4 presents a complete description of the *RDP* protocol, and then Section 5 gives the results of the simulation study.

2 Characterizing data streams

Real time data, such as voice and video, have predictable characteristics and make specific demands of the networks that must transfer it. Data can be transmitted in packets of a constant size that are produced at a constant rate or, in some cases, the bandwidth may vary, due either to variable packet size or rate, with a predefined maximum, and perhaps a non-zero minimum. The variation may also be predictable based on a model of how the data is generated. Depending on the equipment used to generate the data, the packet size and rate may be negotiable. Certain applications, such as voice, produce packets at the given rate only for a fraction of the time ([4, 12]).
The ON-OFF model[13, 14] is used to characterize a data stream. In this model two distinct periods are considered. During the ON period packets arrive deterministically at intervals of T milliseconds. No packets arrive during the OFF period. The ON and OFF periods alternate and are distribute exponentially with rates α and β respectively. The inter-arrival distribution for this process is given by

$$F(t) = [(1 - \alpha T) + \alpha T(1 - EXP(-\beta(t - T)))] * U(t - T)$$

where $U(t)$ is a step function. The arrival rate λ is given by:

$$\lambda = \frac{\beta}{T(\alpha + \beta)}$$

Four parameters are used:

R_{max}	:	maximum packet's arrival rate (i.e. $1/T$) $[pkt/sec]$;
T_{ON}	:	average length of the ON period (i.e. $1/\alpha$) $[sec]$;
T_{OFF}	:	average length of the OFF period (i.e. $1/\beta$) $[sec]$;
s_{max}	:	maximum packet size $[bytes]$.

A constraint has been added to make data streams more controllable. The limitation is on the values that the average length of the OFF period can have. The relation:

$$\frac{1}{R_{max}} < T_{OFF}$$

has to hold, otherwise, due to the hypothesis of exponential distribution of the ON and OFF periods values, it wouldn't be possible to determine whether a pause in the transmission is caused by an inter-packet delay or by a switching between ON and OFF periods.
The average arrival rate is given by:

$$R_{ave} = \frac{R_{max} T_{ON}}{T_{ON} + T_{OFF}}$$

The long term data rate is:

$$R_{ave} * s_{max}$$

The used burstiness index is:

$$\frac{R_{ave}}{R_{max}} = \frac{T_{ON}}{T_{ON} + T_{OFF}}$$

At any time interval of length Δt, the maximum number of packets arriving at the node's interface are:

$$R_{max} * \Delta t$$

The amount of buffer space required is:

$$s_{max} * R_{max} * \Delta t$$

3 IP'–based nodes

The Internet Protocol software executes as a single, self-contained process and a set of network interface queues is used to send and receive datagrams.
The network interface defines the interaction between the protocol software in the operating system and the underlying hardware. It hides hardware details and allows protocol software to interact with a variety of network hardware using the same data structures.
The operating system uses *device driver* software to communicate with hardware I/O devices. The devices' drivers don't call IP directly, instead queues are used to synchronize communication. There is an input queue associated with each network device. IP repeatedly extracts a datagram from one of the queues, uses a routing table to choose a next hop for the datagram and sends the datagram to the appropriate output network interface for transmission or to a higher-level protocol on the local machine.
A typical IP node implementation can be modeled as shown in Figure 1(left). The four stations describe the delays introduced from the process: to move the packet from the input queue to memory, to execute computation which is packet size dependent (PSD), to execute computation which is packet size independent (PSI) and finally to copy the packet back to the output queue.

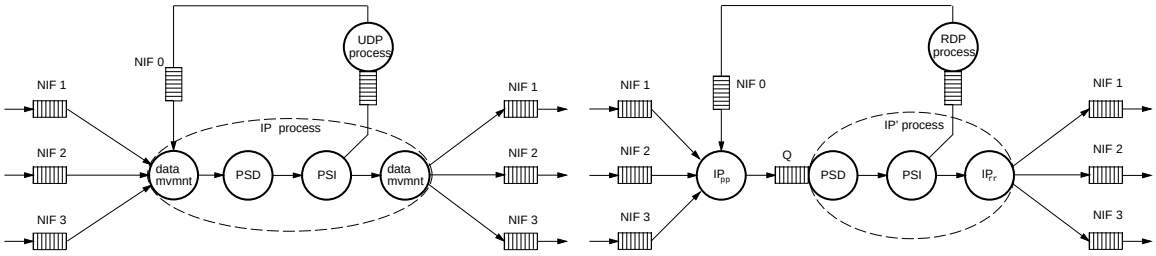

Figure 1: IP node (left) and IP' node (right).

In the IP' node, Fig 1(right), two separate processes are used, namely the IP_{pp} (IP preprocessor) and the IP'.
The IP_{pp} process copies packets into the internal queue Q. While doing this operation, it evaluates their priority testing the `protocol` field of the IP packet header. Two priority levels are used: high for real time connections (RTC) and low for the best-effort connections (BEC, the typical Internet traffic). If a packet belongs to a RTC, further computation is executed. The content of required field `Op` in the transport protocol header is taken and resources handling operations are performed (see Section 4 for the details).

The second process, the IP' process, executes all the original IP protocol operations. The packet size dependent and independent operations are executed as in an IP node with only one input network interface, grabbing packets from the internal queue according to their priority. The element called IP_{rr} represents the IP re-router. Other than copying the packet to the output queue, it will, if required, force the RTC packets to use the path defined during connection establishment. This operation ensures that all RTC packets will always follow a unique route. Best-effort connections packets are just sent ahead toward the port the IP process determined (details in Section 4).
A characteristic that would be desirable for output NIF queues is to give higher priority to RTC packets. This would avoid an overloaded output interface to drop high priority packets before low priority packets. This work won't consider the details of this modification.

3.1 Node bandwidth

The per-packet *service time* of an IP' node is described using[15, 16]:

$$S(ps) = (k_m + k_d) * ps + k_i$$

where:

- ps is the size of the packet in bytes;
- k_m represents the amount of time required to move (copy) the incoming packet from the network interfaces to the OS kernel memory or vice-versa;
- k_d represents the amount of time required to process the incoming packet which depends on the packet size. It can be considered mainly deriving by the checksum computation time.
- k_i represents the processing time which is independent from the packet size. It can be considered mainly deriving by the routing table lookup time.

For any given value of ps it is possible to derive the maximum service rate (i.e., bandwidth) that can be provided. This value is given by:

$$BW(ps) = \frac{ps}{S(ps)} \ [byte/sec].$$

This is a monotonically growing function with maximum value (which coincides with the maximum theoretical node bandwidth), equal to:

$$BW_{MAX} = \lim_{ps \to \infty} BW(ps) = \lim_{ps \to \infty} \frac{ps}{(k_m + k_d) * ps + k_i} = \frac{1}{(k_m + k_d)}$$

At any given instant t, the maximum bandwidth that the node can provide is:

$$BW_{max}(t) = BW(ps_{min});$$

where ps_{min} is the least value of the packet sizes of all connections that are being controlled.

3.2 Node status data

Seven variables are used by the resources handling algorithm. These are:

- P indicates the least value of the packet sizes of all connections currently being controlled.
- B is the amount of *reserved* bandwidth in bytes per second;
- U is the amount of bandwidth currently in use;
- N_Q is the size of the internal buffer in bytes.
- Q_H is the maximum amount of the internal buffer memory that can be used by real-time connections.
- L_H, L_L are the amount of internal buffer memory currently in use by RTC packets and by BEC packets respectively.

P is initialized to ∞; this means that at start-up we are assuming that we can accept requests for bandwidth up to BW_{MAX}. B and U are initialized to 0. At any instant t the still available bandwidth is given by: $BW(P) - (U + B)$. Q_H, L_H and L_L are initialized to 0.
Other then this data a table called `RTCtable` is present in each node and is shared between the IP_{pp} and the IP_{rr} processes.
The table's components are records with six fields. The first field contains the connection identification, the second field contains the requested (utilized or simply reserved) bandwidth, the third field contains the requested (utilized or reserved) amount of internal buffer, the fourth field the maximum size of the packets, the fifth field is used to distinguish between a reservation and a confirmed reservation and the last field contains the used output NIF.

```
struct RTCrecord {
    char    id[12];
    integer bw;
    integer bmem;
    integer pktsz;
    boolean confirmed;
    char    outputNIF;
};
```

The connection identification (`id`) is obtained using the source IP address, the destination IP address, the source port and the destination port put back to back one with the other. The obtained value is ensured to be unique for each connection and can always be built either from the IP_{pp} and from the IP_{rr} using the IP header and the transport protocol header.

3.3 Managing resources

Beside the necessary basic functions to handle the incoming packets, like determining the type of connection (RTC or BEC), giving priority to the packets, generating the connection identification (`id`) or handling the `RTCtable` , the IP_{pp} module provide functions to manage resources (i.e., *bandwidth* and *internal buffer*). These functions are used to reserve resources or to make them available to other connections.
The functions are: `reserve`, `confirm` and `release`.

Function `reserve`

Three values are passed to this function, namely `id`, `ps` and bw_R indicating respectively the connection identification, the connection packet's size and the requested bandwidth (in bytes per second).
The function reserves, if possible, the requested amount of bandwidth. In case not enough bandwidth is available the amount left is allocated.
The detailed code is shown in fig 2. First thing that is checked is whether there is already a reservation done for connection `id`, in which case the old reservation has to be deleted. Then, after controlling if the value of P has to be updated the amount of available bandwidth (variable `A`) is computed. A comparison with the requested amount is done to verify if all the available bandwidth has to be given or only a fraction of it. After this, variable B is updated and the new element in `RTCtable` is created. The function returns the reserved amount value.

```
function reserve (id:octectString; ps, bw_R:integer) : integer;
begin
   if (exist (id, x_1, x_2, x_3, x_4, x_5) in RTCtable ) then begin
      B := B - x_1;
      RTCtable := RTCtable - (id, x_1, x_2, x_3, x_4, x_5 );
   end
   if ( ps < P )
      P := ps;
   A := BW(P)-(U+B);
   if ( A ≥ bw_R )
      reserve := bw_R;
   else
      reserve := A;
   B := B + reserve;
   RTCtable := RTCtable + (id, reserve, 0, ps, FALSE, 0);
end
```

Figure 2: IP_{pp} reserve code.

At this point the fraction on bandwidth reserved is no more available to other connections. The reservation has anyway to be confirmed using the function `confirm`.

Function `confirm`

This function makes permanent the bandwidth reservation and makes the reservation of the internal buffer. To fairly assign buffer, each connection is given a quantity which is proportional to the amount of bandwidth reserved. To decide how much buffer is to be assigned the following relation, which holds at each time and in each node, is used:

$$\sum_{i=1}^{n} BW_i \leq BW(P)$$

where BW_i is the amount of bandwidth reserved by connection i and n is the number of controlled connections.
So, for example, connection j, which has reserved BW_j bytes per second, will have $BW_j/BW(P)*$ N_Q bytes assigned.
The code of function `confirm` is shown in fig 3.

```
function confirm ( id:octectString ) :  integer;
begin
   get (x0, x1, x2, x3, x4, x5) ∈ RTCtable | x0=id
   B := B - x1; { Reserved bandwidth ...}
   U := U + x1; { ... becomes in-use bandwidth.  }
   x2 := x1/BW(P)*NQ;
   QH := QH + x2;
   RTCtable := RTCtable + (x0, x1, x2, x3, TRUE, x5);
end
```

Figure 3: IP_{pp} confirm code.

Function `release`

This function makes the bandwidth and the internal buffer memory available to other connections. After checking whether the resources were actually used or only reserved, the status variables are updated, the entry in the `RTCtable` is deleted and a new value for P is computed. The code is shown in fig 4.

3.4 Managing memory

Before adding a new packet to the internal buffer IP_{pp} has to check whether this operation would generate overflow.
With RTC (high priority) packets the following relations is checked: $Q_H - (L_H + ps)$; where L_H keeps the current amount of buffer used from these packets. If the computed value is greater than or equal to 0 the packet is copied, otherwise the packet is discarded. With RTC control packets a looser limit is used, allowing IP_{pp} to "steal" memory destined to BEC packets. The test used becomes: $N_Q - (L_H + L_L)$.

```
function release ( id:octectString ) :  integer;
begin
   get (x0, x1, x2, x3, x4, x5) ∈ RTCtable | x0=id
   if ( x4 = FALSE ) begin
      B := B - x1;
   end else begin
      U := U - x1;
      QH := QH - x2;
   end
   RTCtable := RTCtable - (x0, x1, x2, x3, TRUE, x5);
   P := min{ pktsz | pktsz ∈ RTCtable };
end
```

Figure 4: IP_{pp} release code.

With BEC (low priority) packets check is done using the relation: $(N_Q - Q_H) - (L_L + ps)$ and again packets are discarded if the computed value is less than 0.
IP_{rr} takes care to decrement the values of L_H and of L_L.

4 The *RDP* protocol

To show how IP' nodes work and how they can be used to obtain higher performances from Internet, an example transport layer protocol is presented. A connection oriented approach is used; an *handshake* phase is present, during which the path between source and destination is established and parameters are defined.
Two machines are involved in a connection: a *client* (or destination) *machine* (CM) and a *server* (or source) *machine* (SM). *RDP* is a client initiated protocol.
Six primitives are defined :

1) `RDP_rtc.request`: Used by a CM to establish a real time connection with a SM. Desired parameters' values are specified.
2) `RDP_rtc.confirm`: Used from a SM to confirm the possibility to open an RTC with the specified parameters' values.
3) `RDP_rtc.propose`: Used from a SM to inform the CM about the available resources on the path.
4) `RDP_rtc.abort`: Used from either the CM and the SM to abort a RTC creation.
5) `RDP_rtc.data`: Used from either the CM and the SM to send data.
6) `RDP_rtc.close`: Used from either the CM and the SM to close a RTC.

The typical signaling scenario is shown in Fig 5.a. The CM sends a `request` to establish a connection with the SM, specifying the required values for the parameters. Nodes along the

path evaluate whether the requests can be completely fulfilled. The SM responds with the `confirm` message which precedes the data sending. The CM is the one that usually closes the connection.
If during connection establishment some node along the path determines that the requests cannot be fulfilled, notification is put in the `request` packet. Once the SM discovers the problem, it informs the DM about the maximum possible availability on the path with a `propose` message.
Two situations can now occur, either the CM accepts the new conditions (case 5.b) and a new connection `request` message is sent or the RTC is dropped (case 5.c).
If the SM finds out that the connection cannot take place at all, it'll issue the `abort` message directly (case 5.d).

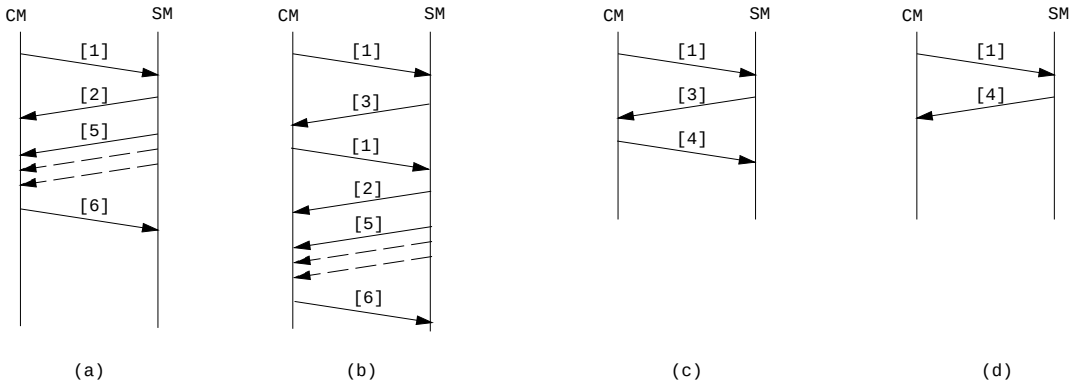

Figure 5: Connection phases.

Packet's format

IP packet structure doesn't need to be modified to handle *RDP* and there is also no need for packets' encapsulation. The number 117 is proposed as a new value for field `protocol` to identify *RDP*[17].
RDP packets' structure is presented in Figure 6. IP_{pp} also checks a part of the *RDP* packets header, looking at the value of field `Op` to determine whether it has to `reserve` resources (`Op` $= 01_{\text{b}}$), to `confirm` resources reservation (`Op` $= 10_{\text{b}}$) or to `release` resources (`Op` $= 11_{\text{b}}$). When `Op` $= 00_{\text{b}}$, IP_{pp} won't do anything.
The field `Type` characterizes the use of the packets and the meaning of the following fields for the *RDP* protocol. Six types are defined, to be represented with the integers from 1 to 6, to indicate: `request`, `confirm`, `propose`, `abort`, `data` and `close` messages.
The field `Max Packet Size` contains the value of s_{max}, while field `Packet Length` contains the actual length of each packet. Fields `Source Port`, `Destination Port` and `Checksum` have the same use as in the *UDP* protocol[18]. Structure 6.a is used by `request` and `propose`. Messages `confirm`, `abort`, `close` and `data` use structure 6.b. The first three of them are actually a particular case of a `data` message, where the field `Packet Length` is equal to 12 and the `data` part is not used.

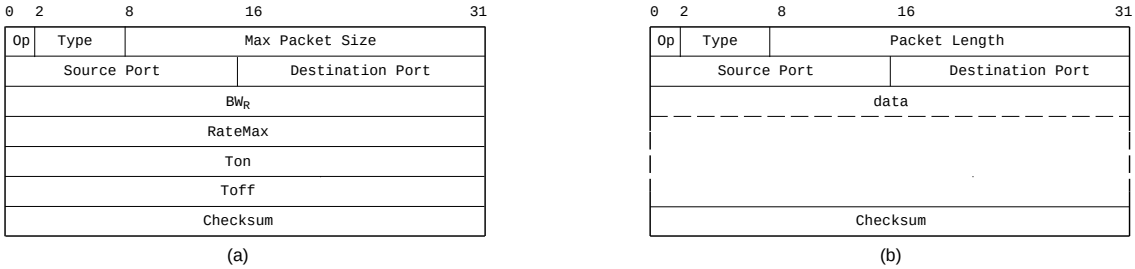

Figure 6: *RDP* packet

4.1 The *RDP* signaling

The RDP_rtc.request message

The client machine uses this messages (fields `Type`= 1 and `Op`= 01_b) to establish a connection. The packet to be sent to the server machine is build setting:
`Max Packet Size`= s_{max}, $\texttt{bw}_\texttt{R} = R_{max} * s_{max}$, `RateMax`= R_{max}, $\texttt{T}_\texttt{on}$= T_{ON} and $\texttt{T}_\texttt{off}$= T_{OFF}.

Since `Op`=01_b the IP_{pp} of each intermediate node will call the `reserve` function. The returned value is placed in the $\texttt{bw}_\texttt{R}$ field. The IP_{rr} process has to put in the `outputNIF` field of the `RTCtable` new element, the value determined from the IP process. This value will be used in the following, to make the packets always use the same route.
The destination server machine, receiving a `request`, checks whether the connection can take place with the requested parameters. If this condition is verified a `confirm` packet is sent followed by the data packets.
If the connection requests can only be partially fulfilled ($\texttt{RateMax} > \texttt{bw}_\texttt{R}/\texttt{MaxPacketSize} \geq R_{ave}$) [1] the `propose` message is sent with field $\texttt{RateMax} = \texttt{bw}_\texttt{R}$ (the value of the others fields remains unchanged).
If the connection requests cannot even be partially fulfilled ($\texttt{bw}_\texttt{R}$ /`MaxPacketSize` $< R_{ave}$) the `abort` messages is sent. See code in Fig. 7 for a detailed description.

The RDP_rtc.confirm message

The server machine uses this message (fields `Type`= 2 and `Op`= 10_b) to inform the CM that the connection can take place.
The IP_{pp} of each intermediate node receiving this message will call the function `confirm` to make the reservation permanent.

The RDP_rtc.propose message

The `propose` message (fields `Type`= 3 and `Op`= 01_b) is used from the SM to try to define new parameters for the RTC, that could be accepted from the network.

[1] R_{ave}=`RateMax*(Ton/(Ton+Toff))`

```
if ( RateMax > (bw_R/MaxPacketSize) ) begin
   if ( (bw_R /MaxPacketSize) ≥ R_ave )
     RDP_rtc.propose;
   else
     RDP_rtc.abort;
else
   RDP_rtc.confirm;
end
```

Figure 7: Server's `request` code

If the CM accepts the new values for parameter R_{max} which is hold in the $\mathtt{BW_R}$ field, a new `request` packet is sent, otherwise the connection is `aborted`, and the reserved resources are released.
Intermediate nodes, handling this message, have probably to decrease the reserved capacity, to match the maximum capacity available on the path. The $\mathrm{IP_{pp}}$'s `reserve` function takes care of the updating.

The `RDP_rtc.abort` message

The abort message (field `Type`= 4 and `Op`= 11_b) is used to release the booked resources on the path on which an RTC was tried to be established.
Nodes receiving an `abort` packet use $\mathrm{IP_{pp}}$'s function `release` to execute the operation.

The `RDP_rtc.close` message

The close message (field `Type`= 6 and `Op`= 11_b) is used to close a connection releasing the resources used on the path.
Nodes receiving a `close` packet call the $\mathrm{IP_{pp}}$'s function `release` to make resources free.

The `RDP_rtc.data` messages

Data packets are distinguished from having field `Type`= 5 and field `Op`= 00_b;
$\mathrm{IP_{pp}}$ in each node, while loading the packet in memory, looks for the relative entry in the `RTCtable` . If the packet is part of a known RTC connection, high priority is given to the packet.
If the reserved area on the internal buffer cannot hold the packet (has should not happen) the node tries to "steal" memory from the not reserved area. In the worst case, the packet is dropped.
The $\mathrm{IP_{rr}}$ finally redirect the packet to the predefined output NIF.

5 Simulation results

A model of the IP' based nodes was designed to analyze its behavior and to measure the performance of the *RDP* protocol. The model is based on the implementation described in [19].
The queuing network structure of a node is shown in Fig. 8. In this case a node with 3 I/O NIF is shown.

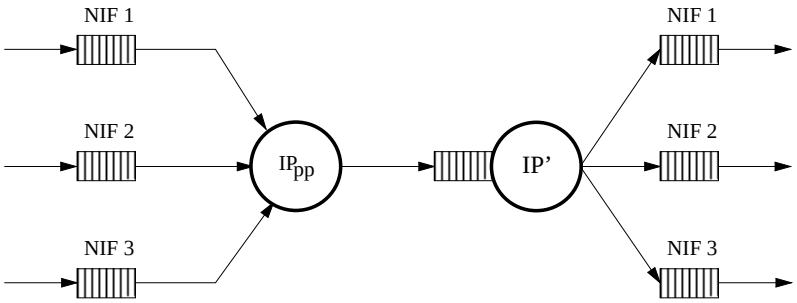

Figure 8: Structure of the node's model.

As seen in Section 3.1 the node service time is modeled using the three parameters k_m, k_d and k_i. The values used were obtained from [15] and are valid for $s_{max} \geq 512$ *bytes*. They are: $k_m = 341 * 10^{-6}$ $[msec]$; $k_d = 183 * 10^{-6}$ $[msec]$; $k_i = 450 * 10^{-3}$ $[msec]$. The value of the service rate is then given from the following function:

$$BW(ps) = \frac{ps}{(524 * 10^{-6}) * ps + 450 * 10^{-3}} \ [byte/sec]$$

with a maximum value of $BW_{MAX} = 1.908\ MBps$.

The network simulated is shown in Figure 9. Four sources were present, each one capable

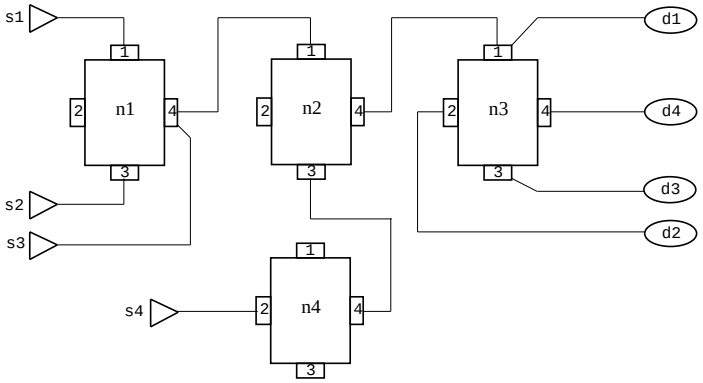

Figure 9: Test network.

to generate real-time and best-effort connections. The approach used to produce bursty

source was based on the ON-OFF model. As described before, this model presents two distinct periods: the ON period, during which packets arrive deterministically at intervals of T milliseconds, and a OFF period, during which no packet arrive. The ON and OFF periods alternate and are distribute exponentially. All sources, either generating a BEC or a RTC, were given the same attributes R_{max}, T_{ON}, T_{OFF} and s_{max}.

Delays analysis

The first goal of the simulations was to understand the behavior of packets' delay.
Four sources were used, three of them (s_1, s_2 and s_4) establishing a RTC and one of them (s_3) establishing a BEC. A period of 10^5 ms was simulated and, while s_1 and s_2 were started at simulation time 0, s_3 was started at simulation time 10000 and s_4 at simulation time 20000 (to verify that a RTC is actually handled with higher priority than a BEC.)
R_{max} was 124 pkt/sec and $s_{max} = 4096$ bytes, so that the four connections together were enough to overload the node.
While N_Q was fixed to $20K$, different combination of the T_{ON} and T_{OFF} parameters were used:

	case a	case b	case c	case d
T_{ON}	350ms	500ms	650ms	800ms
T_{OFF}	650ms	500ms	350ms	200ms

Figures 10, 11 show some of the results. As can be seen, the real time connections present

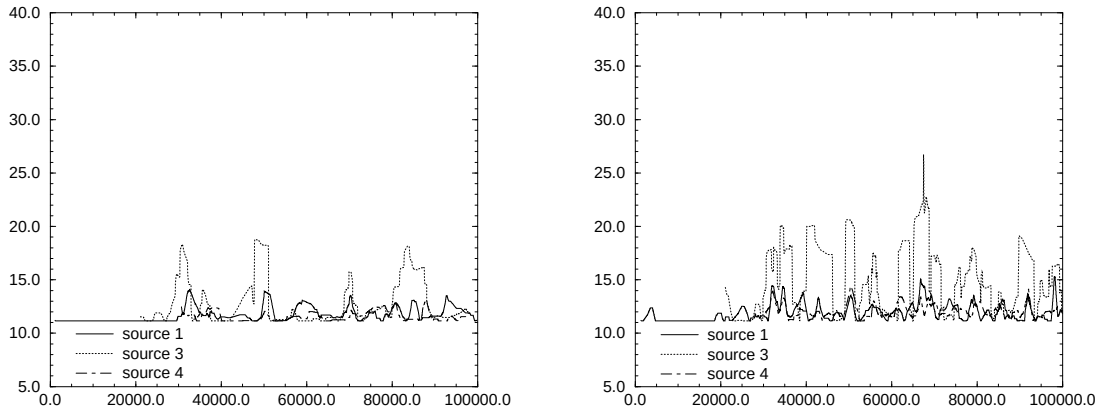

Figure 10: Cases a and b.

a much more stable behavior than the BE connection. The delays of the packets stays in fact between a close range. Moreover, as the table below shows, the values of the RTC present a standard deviation smaller then the BEC. This result show that is now possible for the network to provide bounded delay for RTC.

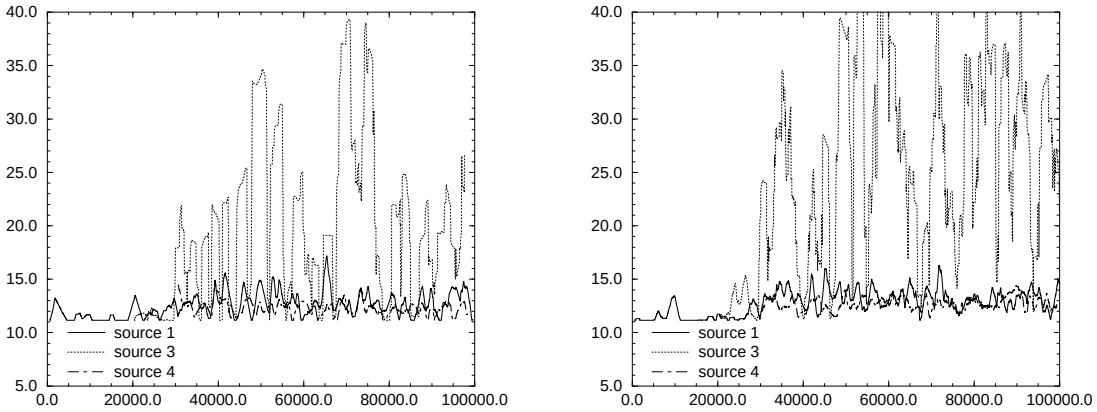

Figure 11: Cases c and d.

	case a		case b		case c		case d	
	$\bar{x}$	$\hat{x}$	$\bar{x}$	$\hat{x}$	$\bar{x}$	$\hat{x}$	$\bar{x}$	$\hat{x}$
source 1	11.727	1.116	11.943	1.340	12.523	1.596	12.789	1.663
source 3	12.473	7.782	14.176	15.222	17.216	24.748	21.778	32.135
source 4	11.617	0.893	12.087	1.131	12.303	1.143	12.706	1.130

Throughput analysis

Throughput for a real time connection is slightly increased and presents, like the delays, a more stable behavior. The data in the plots are obtained using a running average of size 1000. (Fig 12). Figure 12 shows a comparison of the behavior of the average and the

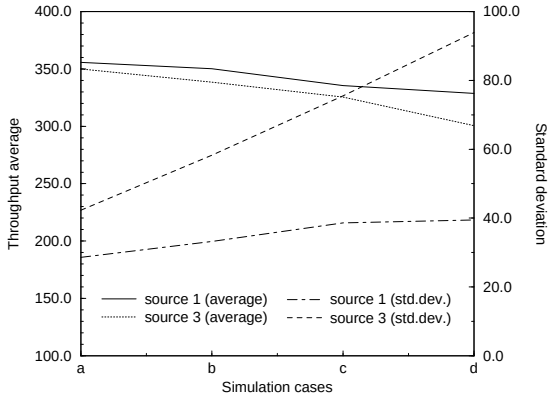

Figure 12: Throughput behavior comparison.

standard deviation for the throughput values from case a through case d. Despite the almost

similar trend for the average value, the standard deviation shows that the RTC presents a more stable behavior. This allows the network to provide bounded throughput for RTC.

Packet loss

Figure 13 shows the behavior of the percentage of packet loss for the BEC through the four cases. Case d presents a loss of more than 37% of packets, whereas the three RTC together, present a loss of 0.04% of the total amount of packets.

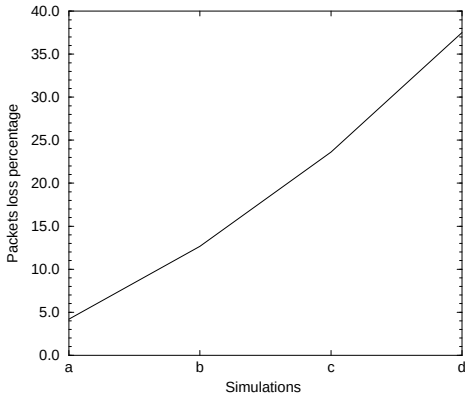

Figure 13: BE connection packets loss.

Queue length

To understand the behavior of the internal queue, simulations related to the previous case *d* were executed again setting $N_Q = 100K$ and making available to the real time connection only a proportion k of the internal buffer. Values used were $k = 25\%, 50\%, 75\%$. Figure 14 shows the percentage of the elapsed time spent by the internal queue of node n_2 with the length specified on the x-axis.

6 Conclusions and Future Work

This paper presented a modification of IP based nodes to allow a simple management of the node's resources. The modification are minimal and not intrusive on the regular Internet data traffic. The new nodes are called IP' nodes.
A protocol called *RDP* was also presented to illustrate a possible structure of a user-oriented protocol that wants to take advantage of the IP' nodes. *RDP* obtains higher performance from the network using the capabilities of the IP' nodes either to allocate resources to a connection and to make each connection follow a unique path. This last feature could be more effective with the associated use of improved routing table update mechanism (e.g., see [20]), in order to provide each node with the most updated information about the network load.

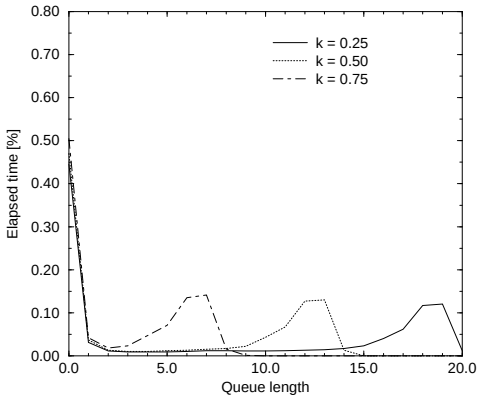

Figure 14: Node n_2 internal queue.

Simulations results were presented to show the behavior of the protocol in different situation and with different values for the characterizing parameters. The simulations showed that this approach can actually bound the variation of delay and throughput. They also show that the number of packets lost can be controlled.
There are still various important points to be investigated. For example, a way to compute the size of the internal buffer required by IP' nodes should be evaluated. Also, a better understanding of the impact of the I/O NIF queue service order would be desirable. Most of these aspects require an analyses of the behavior of data streams coming for example from video transmissions, to see how the ON-OFF model can completely and accurately describe their behavior.
The *RDP* protocol can be improved, too. *Policing* algorithms could be inserted so that, when an attempt to violate the requests is detected, corrections can be taken. For instance if a source tries to send data with a rate higher than the declared R_{max}, an approach like the *leaky bucket* can be used [21]. Also a way for *RDP* protocol to provide the user application with some feedback information about the packets delay and throughput variation would be useful.

Acknowledgments

I would like to thank Riccardo Bettati for his indications and suggestions.

References

[1] D. Ferrari, "Client requirements for real-time communications service," *IEEE Communications Magazine*, vol. 28, pp. 65–72, November 1990.

[2] V. Jacobson, "Congestion avoidance and control," *Proceedings of SIGCOMM 88, Stanford, CA, USA*, pp. 314–329, August 1988.

[3] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE Transactions on Networking*, vol. 1, pp. 397–413, August 1993.

[4] D. Anderson, R. Herrtwich, and C. Schaefer, "SRP: A resource reservation protocol for guaranteed-performance communication in the Internet," Tech. Rep. 90-006, ICSI-International Computer Science Institute, Berkeley, CA, February 1990.

[5] H. Schultzrinne and S. Casner, "RTP: A real-time transport protocol." IETF, July 1993.

[6] C. Topolcic, "Experimental internet stream protocol, version 2 (ST-II)." RFC 1190, October 1990.

[7] D. Comer and R. Yavatkar, "FLOWS: Performance guarantees in best effort delivery systems," *Proc. of the 8th IEEE INFOCOM'89, Ottawa, Ont., Canada*, April 1989.

[8] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A new resource reservation protocol," *IEEE Network*, vol. 7, September 1993.

[9] D. Ferrari, A. Banerjea, and H. Zhang, "Network support for multimedia - a discussion of the Tenet approach," Tech. Rep. 92-072, ICSI-International Computer Science Institute, Berkeley, CA, October 1992.

[10] D. Verma and H. Zhang, "Design document for RTIP/RMTP." ICSI-International Computer Science Institute, Berkeley, CA, 1994.

[11] P. Di Genova, "Real-time communication on computer networks: Introduction and analysis of the Tenet approach of Berkeley," Master's thesis, Tesi di Laurea - University of Bologna, Bologna - Italy, December 1993.

[12] D. Ferrari, "Real-time communication in packet-switching wide-area networks," Tech. Rep. 89-022, ICSI-International Computer Science Institute, Berkeley, CA, May 1989.

[13] H. Yamada and S. Sumita, "A traffic measurement method and its application for cell loss probability estimation in ATM networks," *IEEE Journal on Sel. Area in Comm.*, vol. 9, April 1991.

[14] H. Akimaru, T. Okuda, and K. Nagai, "A simplified performance evaluation for bursty multiclass traffic in ATM networks," *IEEE Transactions on Communications*, vol. 42, May 1994.

[15] J. Kay and J. Pasquale, "A performance analysis of TCP/IP and UDP/IP networking software for the DECstation 5000," Tech. Rep. 92/18, SEQUOIA 2000, 1992.

[16] D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An analysis of TCP processing overhead," *IEEE Communications Magazine*, pp. 23–29, June 1989.

[17] J. Reynold and J. Postel, "Assigned numbers." RFC 1340, July 1992.

[18] J. Postel, "User datagram protocol." RFC 768, September 1980.

[19] D.E. Comer and D.L. Stevens, *Internetworking with TCP/IP - Vol. II. Design, Implementation and Internals*. Prentice Hall, 1991.

[20] S. Deering, "ICMP router discovery messages." RFC 1256, September 1991.

[21] N. Yin and M.G.Hluckyj, "Analysis of the leaky bucket algorithm for ON-OFF data sources," *Journal of High Speed Networks*, vol. 2, no. 1, pp. 81–98, 1993.