Adding Dynamic Behaviour To Domains: Combination of In Band Implicit Signalling and Policy Management

PAOLO DI FRANCESCO, PIERLUIGI GALLO, VINCENZO FODERA'
CRES, Centro per la Ricerca Elettronica in Sicilia
Via Regione Siciliana 49
90046 Monreale, Palermo,
ITALY

CORINNE SIEUX, HERVÉ AIACHE
THALES Communications
Advanced Information Technologies Dept.
BG COM/TCF/TSS/TAI Dept.
160, bd de Valmy – BP 82 – Immeuble Cook
92 704 Colombes Cedex
FRANCE

Abstract: - Network management becomes complex and hard to understand. Network administrators have now to deal with new data patterns and advanced control services to support ISPs’ customised commercial offers. The complexity of data patterns is related to new needs: the users ask for more new services to ISPs (e.g. peer2peer scenarios are opening up new traffic patterns) or advanced properties to the legacy services. For example, a videoconference session could need new enhanced properties, like QoS support, to enhance users’ needs. Statefull protocols, (e.g. RSVP) do not scale very well in the core network, and stateless routers supporting DiffServ are used to handle the huge amount of connections per second. Taking into account the complexity of these elements, we show how the combination of a non conventional use of AF-DiffServ queue disciplines supporting end-to-end QoS and an enhanced Policy architecture, provide to network administrators the ability to control a device or a domain, using only few understandable parameters.

Key-Words: - DiffServ, Policy, QoS, Network Management

1 Introduction

Nowadays the ISPs are evaluating the introduction of new multimedia contents in the Internet. Unfortunately, the more the contents depend on stringent constrains (delay, bandwidth, etc.) the more difficult is for the ISP to provide services. While the statefull approach has been recognized as not scalable, most of the stateless routers implementing queue disciplines (e.g. RED or RIO) have been proved, in simulation, to perform poorly with real time UDP traffic [18], [12]. Extensions or improvements of the “plain” RED mechanism have been proposed. In [19], the RED mechanism has been extended providing to the running flows a fairer usage of the available bandwidth when congestion occurs: a more selective discard method is implemented to protect adaptive flows from non-adaptive flows. The idea is further explored in [22] where better performance with UDP flows is achieved. Unfortunately, these mechanisms have not proved to provide a congestion avoidance solution. For this reason, the GRIP mechanism [13] has been proposed to support the QoS (Quality of Service) on UDP data flows, and it has been proved to have good performances in simulation environments. In this paper we show, via experimental data collecting, that another interesting side effect of the introduction of the GRIP mechanism, inside the core routers, is its ability to be administered easily using few understandable parameters via a Policy Graphical User Interface.

In section 2, we introduce the basics about GRIP and policy mechanisms and how the two technologies have been integrated. In section 3 we show some performances comparison, and in section 4 we present some concluding remarks.
2 GRIP and Policy basics

2.1 GRIP - Gauge & Gate Reservation with Independent Probing

Differently from the Best Effort approach, where any host can send packets anytime with the hope of being served by the network, the GRIP approach conforms to the CAC paradigm where the connections are accepted or rejected accordingly to the network status (congested/not-congested). In this latter case the hosts can start sending packets, after the probing phase, only if the connection has been accepted.

The latest works on this topic have evolved the GRIP ability of being integrated with legacy architectures (e.g. RSVP [10] and RTSP [17]) and its DiffServ compliancy has already been presented in [12] showing how AFx1 packets can be thought as data packets, and the AFx2 as probe packets.

In our implementation of the GRIP mechanism, the decision criterion rejects or accepts a packet accordingly to the measures done on the traffic offered to the output link, applying the dropping function shown in Figure 1.

\[ P(T) = \begin{cases} 
0 & \text{if } T < T_{\text{min}} \\
\frac{T - T_{\text{min}}}{T_{\text{max}} - T_{\text{min}}} & \text{if } T_{\text{min}} < T < T_{\text{max}} \\
1 & \text{if } T > T_{\text{max}} 
\end{cases} \]

\[ (1) \]

\[ (2) \quad t = \frac{\text{Traffic offered to the Link}}{\text{Link bandwidth}} \]

2.2 Policy Architecture

The purpose of a Policy Architecture is to provide mechanisms to translate business goals into simple rules (i.e. commands) to be applied on the network devices.

The basic elements of such an architecture are shown in Figure 2 and listed below:

- The Administrator views and expresses requirements at a high level using a Management Application.
- PEP (Policy Enforcement Point). It represents nodes or devices of the network to be managed. It enforces policy rules directly managing device properties such as buffer size, queue disciplines, schedulers and it provides feedback to PDP, informing it of errors, events, etc.
- PDP (Policy Decision Point). It represents a policy server that can install or remove decisions into the PEP. It can be implemented in one or more devices even in the same device where the PEP is present. To take these decisions, it uses high-level policies provided by an administrator or stored in a policy repository. It can support several PEPs.
- Policy Repository. It is a policy storage server that can be used by the PDP for processing its decision. It centralizes data, avoids redundancy and synchronization problems. For example LDAP directory can be used to implement a policy repository.
2.3 GRIP Policy-Based Management Architecture

In the previous sections GRIP and a generic policy architecture has been presented. In Figure 3 the modifications needed to support GRIP over a Policy Management Architecture (i.e. a QoS Policy Driven Architecture) are shown.

![Figure 3 GRIP implemented in a Policy Architecture](image)

To develop this architecture, considering the deep modifications involved in the PDP/PEP, an open and sources available environment was needed. For this reason the Linux Operating System has been used for the software router (see Figure 4) and a Java console has been written from scratch for the PDP/GUI part.

![Figure 4 GRIP PEP implementation details](image)

The GUI (shown in Figure 5) has been logically divided in two main panels:

- **Data View.** In this section the system administrator can select Data or Probe views using a JComboBox. The Data view summarizes the offered traffic: the green line represents the total offered traffic, the red line the discarded traffic and the blue line the served (i.e. not dropped) traffic. In the ideal case the dropped traffic is equal to zero and only a blue line is shown on the screen. Moreover, in our implementation each connection sends only one probe packet, hence monitoring the probe packet diagram is equivalent to monitoring the connection accept/reject process.

- **Control View.** In this view the system administrator can control the accept/reject mechanism varying the two thresholds \((T_{\text{min}}, T_{\text{max}})\) which are the values already introduced in the \(P(T)\) function (see Figure 1). Therefore, the link can be controlled via the use of 2 simple watermarks, a lower threshold and an upper threshold.

- A third section, in the lower part of the window, is also present giving to the user the ability to read the logs, and check if the rule enforcements have been successfully applied.

![Figure 5 The GUI panel](image)

3 Performances Comparison

While the simulation results show a better behaviour of GRIP towards classical queue mechanisms, a software router, which suffers of some Operating System granularity constrains, could perform differently. Moreover in a policy environment where the system administrator has an important role into the decisional process, it is very important to show that the management console, which shows abstract concepts like traffic and thresholds, can apply in the correct mean the policy rules.
In the following sections we will show some performance results that prove the superior GRIP Policy-based administering capabilities.

### 3.1 GRIP Performances

Simulations have shown that GRIP performs well with UDP data flows under some circumstances: the traffic must be GRIP compliant, i.e. a probe packet must be sent to test the link congestion status before the data packets. In this section we show the experimental results obtained with a GRIP Linux software router implementation, which can be controlled by a GUI interface. Figure 6 shows the test-bed used for the measurements done in this work.

![Figure 6 Test-bed topology](image)

The transmitter sends unidirectional CBR 32/64/128Kbit GRIP compliant connections to the receiver. A physical bottleneck is obtained using a 10Mbits link and the offered traffic is 120% of the output link bandwidth (i.e. 12Mbits/sec). The traffic details are shown in Figure 7.

<table>
<thead>
<tr>
<th>GRIP compliant transmitter settings</th>
<th>offered load</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>32kbps</td>
</tr>
<tr>
<td>connection inter-time [ms]</td>
<td>100</td>
</tr>
<tr>
<td>connection duration [ms]</td>
<td>37500</td>
</tr>
<tr>
<td>pkt intertime [ms]</td>
<td>325</td>
</tr>
<tr>
<td>pkt payload [bytes]</td>
<td>1300</td>
</tr>
</tbody>
</table>

![Figure 7 Traffic offered](image)

The table suggests that the best values are obtained with higher $T_{\text{hmax}}$. Unfortunately what the table does not highlight are those cases where fluctuations of the traffic are present. Indeed the average value does not take into account possible negative or positive peaks. In our experience, to avoid this phenomenon better results are obtained with larger intervals, i.e. a smoother $P(T)$ function.

To prove the simplicity of the policy-driven GRIP architecture, we asked to 3 different GRIP unaware testers to set up the GRIP router via the GUI to achieve the 60% link usage under different traffic conditions. We revealed them neither the source traffic data details (connections per second, Kbit/sec for each connection, etc.) nor the mechanism internals and we made the average on 5 minutes traffic after they thought they had found their best setting.

![Figure 9 GRIP GUI driven configuration](image)

Surprisingly some of the administrators were able to achieve better not-fluctuating results than the ones provided in the previous table.

### 3.2 GRIP vs RED

In this section we want to compare the GRIP policy architecture to the RED mechanism using the same test-bed described in the previous section and shown in Figure 6. In particular we want to show that: i) the default parameters found on the Linux RED man page cannot achieve a 60% link utilization and ii) it is not clear how to achieve a path towards better
settings. In [16] a set of parameters for the RED mechanism has been proposed, and in [6] a mechanism for finding more scalable set of parameters has been studied. Unfortunately in both cases, no evaluation of real-time UDP traffic have been explored, hence we used as first step the default values found on the Linux RED man page to understand how they perform under the same test-bed conditions.

In Figure 10 it can be noted that the link occupancy is close to 100%. Indeed, under this scenario, the RED mechanism is not able to limit or reject connections. As a side effect the output link is completely congested and data packets are dropped. Hence, applications that use a differential compression scheme for audio/video data flows will not benefit of any QoS.

Finally (see Figure 11), we tried to achieve the 60% goal choosing another set of RED parameters. In [12] it has been noted that using small packet queues, RED looks to perform better. In the Linux implementation, the queue sizes are not expressed in terms of packets but bytes. Therefore, considering that the packet size is fixed to around 1300 bytes we started from the min=1500 bytes and extrapolated the remaining values accordingly to [16].

As shown in Figure 11 these settings seem more promising and closer to 60%, but a better look reveals that the data dropping percentage is not acceptable.

4 Conclusion and future work
In this paper the performances of a new GRIP policy driven mechanism have been evaluated and compared to RED via a Linux software router test-bed. The results show that the RED queue discipline is not well suited for real-time UDP flows. Conversely, the combination of GRIP and a simple policy GUI allows a better understanding of the per-hop behaviour thanks to the low number of parameters (2 in the presented scenario) required to set up the router.

The encouraging results on the per-hop-behaviour will be extended in our future work to a per-domain-behaviour with the use of a centralized and more intelligent policy manager. Indeed, although our policy based GRIP queue discipline gives to the administrators the ability of controlling the maximum link occupancy within a fixed desired range and consequently providing QoS over static paths, more investigation is still needed when a dynamic routing change occurs (e.g. link failure). Future research work will consist in exploring cooperative and explicit mechanisms between routers, in order to propagate QoS-related information inside the whole domain and eventually enforcing route change to avoid link saturation.

References: