ETSI TR 182 015 V1.1.1 (2006-10)

Technical Report

Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); Next Generation Networks; Architecture for Control of Processing Overload

Reference

DTR/TISPAN-02026-NGN

Keywords

control, architecture

Contents

Intellectual Property Rights

Foreword

This Technical Report (TR) has been produced by ETSI Technical Committee Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN).

Introduction

Next Generation Networks (NGN) are required to provide real-time services such as authentication, location, presence information, user registration, accounting, and separation of bandwidth control from call/session control (as with the gateway control protocol H.248); and these functions will generally be distributed over a number of servers. The protocols used between the servers could comprise any, or all, of the following: SIP, HTTP(S), Radius, Diameter, DNS, COPS, SNMP, SMPP, SAML, LDAP, Parlay, Java, SOAP, Midcom, H.248, SIP-I, INAP, etc.

An analysis of IETF Working Groups suggests that much thought has been, and is being, given to the management of bandwidth and router congestion (see, for example, RFC 3124 [5] "The Congestion Manager"), but almost none to control of host and server processing overload. Indeed, although some of the above listed protocols (e.g. SIP and HTTP) do provide response or status codes that might be used to indicate processing overload, none explicitly specifies an overload control mechanism.

The same conclusion also seems to apply to the following non-IETF protocols: H.323 [8], SOAP, SAML, SMPP, and Parlay.

This contrasts with the telephony/ATM world where there are examples of protocols that have built-in overload control features: INAP, ISUP, BICC, PNNI and more recently overload control package H.248.11 [7]. Such controls are crucial during extremes of network operation where surges of service requests (or Denial of Service attacks) can be an order of magnitude greater than normal demand levels.

As a general rule, an NGNs servers can experience prolonged processing overload under the appropriate circumstances (e.g. partial, or full, server failure, high rates of incoming service requests). Consequently, it needs to be equipped with some form of overload detection and control (including expansive controls such as load balancing and resource replication), in order to keep response times just low enough under such processing overload to preclude customers abandoning their service requests prematurely.

The usual way to provide load control is to build a mechanism into each protocol that needs it (for example ISUP ACC, and the H.248.11 [7] Congestion Control Package). However, rather than building a variety of mechanisms into a range of protocols, established through a number of standards fora, it ought to be quicker and cheaper to solve the problem once, in a way that is independent of the main protocols. This could be done by designing a separate overload control protocol with associated load control functions which together detect processing overload, adapt and distribute restriction levels and apply restriction. This protocol is called GOCAP (Generic Overload Control Application Protocol).

The perceived benefits are:

1) IT- and internet-based protocols will more easily transfer into the service-level assured world of the telecommunications operators.

2) Manufacturers have only one overload control mechanism to implement and maintain.

3) It would simplify the route through standards bodies.

4) It would provide flexibility of implementation (no need to implement if not required).

The present document explores the some of the options for GOCAP and the functional entities required to support its use as an overload control. The specific NGN requirements and actual protocol specification of such an overload control will be described separately.

1 Scope

The present document describes the architectural principles that are required to provide effective control of processing overload in networks compliant to the TISPAN NGN Architecture. As such it constitutes a discussion of the requirements for the protocols required to support the NGN overload control architecture.

The scope is limited to the control of processing overload at NGN processing resources caused by service requests coming from session-based or command-response applications by controlling the rate at which those applications send service requests to an overloaded resource. It does not extend to the overload of transmission bandwidth whether used for the user plane or for the control plane.

2 References

For the purposes of this Technical Report (TR), the following references apply:

NOTE: While any hyperlinks included in this clause were valid at the time of publication ETSI cannot guarantee their long term validity.

[1] ETSI ES 283 039-3: "Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); Specification of protocols required to support the NGN Overload Control Architecture; Part 3: Overload and Congestion Control for H.248 MG/MGC".

[2] Whitehead M J and Williams P M, "Adaptive Network Overload Controls", BT Technology Journal, Vol. 20, No. 3, July 2002.

[3] ITU-T Recommendation E.412 (01/2003): "Network management controls".

[4] ETSI TR 180 001: "Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); NGN Release 1; Release definition".

[5] IETF RFC 3124: "The Congestion Manager".

[6] ITU-T Recommendation H.248.10: "Gateway control protocol: Media gateway resource congestion handling package".

[7] ITU-T Recommendation H.248.11: "Gateway control protocol: Media gateway overload control package".

[8] ITU-T Recommendation H.323: "Packet-based multimedia communications systems".

[9] ITU-T Recommendation H.248.1: "Gateway control protocol: Version 3".

[10] ITU-T Recommendation E.164: "The international public telecommunication numbering plan".

[11] ETSI EN 383 001: "Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); Interworking between Session Initiation Protocol (SIP) and Bearer Independent Call Control (BICC) Protocol or ISDN User Part (ISUP) [ITU-T Recommendation Q.1912.5, modified]".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

admission control: control that accepts or rejects request on the basis of system load state

NOTE: This "admission control" relates only to requests accepted and rejected on the basis load state of processing resource. This is a separate control to that used to manage access to transport resource.

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

A	Admission (control, function)
ACC	Automatic Congestion Control
ACL	Access Control List
ADC	Automatic Destination Control
AF	Application Function
AMG	Access Media Gateway
API	Application Programming Interface
A-RACF	Access Resource Admission Contol Function
ATM	Asynchronous Transfer Mode
BGF	Border Gateway Function
BICC	Bearer Independent Call Control
C	Communication (applications)
C	Control (variable)
CA	Call Agent
COPS	Common Open Policy Service
D	Distribution (function)
DNS	Domain Name System
DoS	Denial of Service
GOCAP	Generic Overload Control Application Protocol
HTTP	HyperText Transfer Protocol
IETF	Internet Engineering Task Force
IMS	Internet protocol based Multimedia core network Subsystem
INAP	Intelligent Network Application Protocol
IP	Internet Protocol
ISUP	ISDN User Part
IT	Information Technology
LDAP	Lightweight Directory Access Protocol
M	Monitor (reject Monitor and restriction Mastering function)
MGC	Media Gateway Controller
NGN	Next Generation Network
NOCA	NGN Overload Control Architecture
OSA	Open Service Access
pA	pseudo-Admission
PDU	Packet Data Unit
PNNI	Private Network to Network Interface
PSTN	Public Switched Telephone Network
R	Restrictor, Restriction (method)
RACS	Resource and Admission Control Subsystem
RADIUS	Remote Authentication Dial-in User Service
SAML	Security Access Markup Language
SCP	Service Control Point
SCTP	Stream Contol Transmission Protocol
SIP	Session Initiation Protocol

SIP-I	SIP profile C of EN 383 001 [11]
SLA	Service Level Agreement
SMPP	Short Message Peer to Peer Protocol
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SPDF	Service Policy Decision Function
TCP	Transmission Contol Protocol
TISPAN	Telecommunications and Internet converged Services and Protocols for Advanced Networking
TLS	Transaction Layer Security
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
VoIP	Voice-over-IP

4 Requirements for NGN overload controls

4.1 NGN overload scenarios

4.1.1 Processing overload sizes

Overloads peak at calling rates much greater than the predictable daily profile peak to which the network can be economically dimensioned. Table 1 taken from [2] shows the range of calling rate measurements taken from BTs network (based on 15 minute samples). We can see that overload can exceed 64 times the systematic peak calling rate for six 15 minutes periods a year. While, during such an overload we might expect a large proportion of call attempts to fail, however, it would be unacceptable for the network to fail completely due to processing overload. In particular, the network would prioritize service of emergency traffic and other important streams.

The reality of massive overloads has been demonstrated by data from PSTN networks, how do these overloads occur?

Table 1: Extremes of the calling rate distribution

Calling rate expressed as a multiple of systematic quarter hour peak	Number of quarter hours per year calling rate is exceeded (% of quarter hours)
2	344 (1 %)
4	139 (0,4 %)
8	51 (0,1 %)
16	22 (0,06 %)
32	17 (0,05 %)
64	6 (0,02 %)

4.1.2 Media stimulated events

This is a family of events such as televotes for TV programmes, ticket sales and phone-ins which can all generate high calling rates to particular small ranges of numbers. In [2], it is reported that these events occur with a frequency of several thousand a month. Some of the largest events are televotes stimulated by TV programmes, and such events can have a very rapid onset, with the calling rate increasing at a rate of 4 k calls per second per second over 6 seconds observed in parts of BTs network. Often these events are known about in advance, so steps can be taken to prepare the overload controls. Also they are usually focussed on a small range of destinations, so controls like ADC [3] may help to reject sessions unlikely to succeed early in the call setup process.

4.1.3 Disasters

Disasters, such as major accidents, terrorist attacks or extreme weather, may stimulate overloads, some times focussed on a few destinations (emergency services, information lines etc) or possibly more diffuse with whole regions seeing an increase in call attempts. In the former case, the network operator may have advance notice of a destination before it is advertised to the general public and be able to arm appropriate network controls (these events are similar to media stimulated events, albeit with much less time to prepare the network). The more diffuse overload may be harder to manage as there are no specific destinations that can be used to target the anomalous load and there is no warning. One of the alarming aspects of disaster scenarios is that the operators network may its self be damaged (e.g. by flooding) reducing capacity at the very time that it is exposed to these additional loads.

4.1.4 Network Failures

Significant processing overloads can be generated by network equipment failures. This may be caused by reducing the capacity available (e.g. the loss of a call agent) or by abruptly terminating existing sessions, triggering mass session releases followed by session re-establishment attempts. It is difficult for the network to shed load in these circumstances (the session clear downs have to be processed so that network resources are recovered and end users are not overcharged). These events happen very rapidly, and require very quick response times from automatic controls.

4.1.5 Conclusion

We can see that the NOCA needs to be able to control severe processing overloads, and that these overloads may appear suddenly, with little or no warning. This means that the NOCA needs to be able to detect overloads very quickly and respond without operator intervention.

4.2 Impact of overload on resources and customer behaviour

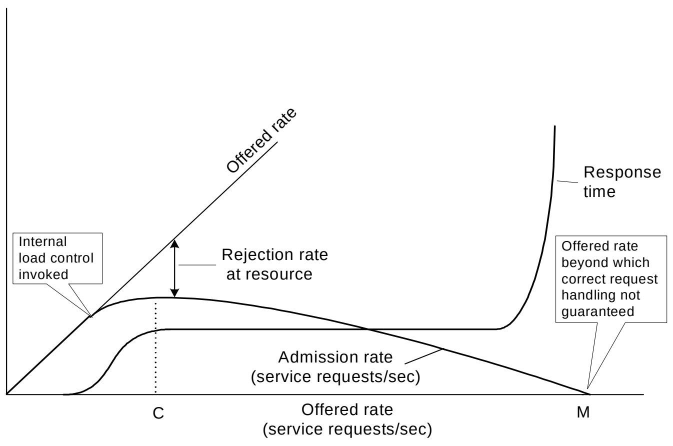

Figure 1: Typical throughput and response time curves for a system with internal load control

Figure 1 illustrates the typical behaviour of a resource that has an internal overload control acting to reject sufficient offered demand so as to keep response times constant, and short, under overload.

Customer persistence can lead to an explosion of repeat attempts when service requests are rejected, resulting in congestion in other parts of the NGN. It is therefore essential to maintain as high as possible effective throughput at an overloaded resource, subject to keeping response time small enough to preclude customers abandoning service requests due to long delays.

Because rejecting fresh calls takes processing effort, effective throughput at an overloaded resource (i.e. admitted service requests/sec) must eventually fall as the load offered to it is increased; and ultimately it will spend all its time rejecting fresh demand. So, to prevent this, it is necessary that controls external to the resource act to reduce the fresh offered load to the level at which its effective throughput is maximized.

4.3 NGN Interfaces requiring overload controls

Figure 2 (taken from [4]) shows the top-level NGN sub-systems (PSTN/ISDN Emulation Subsystem, IP Multimedia Subsystem, etc). The specific overload requirements of the NGN will be documented in the individual functional specifications. The realization of this architecture will require many protocols. Table 2 shows a list of protocols, many of which could be required for the TISPAN NGN, together with their overload control status. Notice that many key protocols are without effective overload control.

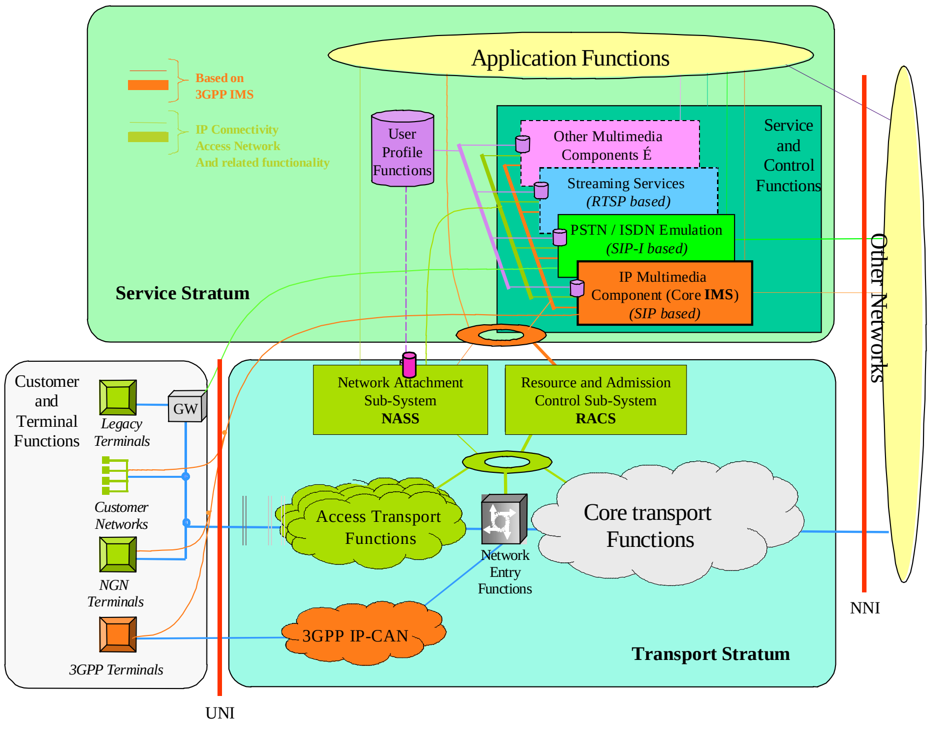
Application Functions
Based on 3GPP IMS
IP Connectivity Access Network And related functionality
User Profile Functions
Other Multimedia Components É
Streaming Services (RTSP based)
PSTN / ISDN Emulation (SIP-I based)
IP Multimedia Component (Core IMS) (SIP based)
Service and Control Functions
Service Stratum
Other Networks
Customer and Terminal Functions
Legacy Terminals
GW
Customer Networks
NGN Terminals
3GPP Terminals
Network Attachment Sub-System NASS
Resource and Admission Control Sub-System RACS
Access Transport Functions
Core transport Functions
Network Entry Functions
3GPP IP-CAN
Transport Stratum
UNI
NNI

Figure 2: TISPAN NGN Overview

Table 2: Overload control status for NGN protocols

Protocol	Description	Has Control ?	Comments
SIP RFC3261	Session control messages, presence etc.	No	Various status codes could be used to drive load control, but SIP does not specify how.
DNS RFC1035	Network host database	No	No DNS message response codes explicitly reserved for server overload. Rcode = 2, "server failure", might be usable.
LDAP RFC3377	Database lookup	No	Result code busy (51) = "server too busy to service the operation". Might be usable to drive an overload control.
HTTP RFC2616	Web based services	No	Various status codes could be used to drive load control, but HTTP does not specify how.
COPS RFC2748		No	Reason code 7 (Insufficient resources) or Error code 4 (Unable to process) might be usable to drive an overload control.
SNMP RFC3411	Management interface	No	The Response-PDU error status value = "resourceUnavailable" might be usable to drive an overload control.
Radius RFC2865	Authentication protocol	No	No provision in Radius for a server (or proxy) to tell a client it is overloaded. Would have to consider use of client time-outs to drive control.
Diameter RFC3588		No	Error code DIAMETER_TOO_BUSY might be usable to drive an overload control.
ISUP	Call control	Yes	Can use ACC.
H.248	Control interface between media gateway and its controller	Partial (MGC to MGW only)	Packages H.248.10 [6] and H.248.11 [7] define overload controls that protect MGW by throttling at MGC. Overload control in the reverse direction (MGW to MGC) is provided by the package ES 283 039-3 [1]. Issue of integration of existing controls with other protocols at MGC.
INAP		Yes	INAP Call Gap message and action defined, but not how to detect overload at an SCP, nor how to adapt the gap interval.

4.4 NGN architectural factors

The NGN overload control architecture needs to take into account the following factors:

- AMG to CA fan-in issue [1]. The possibility of thousands of small AMGs (with, say, 20 to 30 customer lines each) parented on a CA, poses a severe overload control problem for the CA. Special overload control techniques may be needed to protect a CA from very sudden, and very severe, calling rates coming from its dependent AMGs. Reference [1] proposes a new control method, to be incorporated into a new H.248 extension package, to resolve this issue when the H.248 protocol is used.

- As we saw in table 2, many protocols which might be used by an NGN have no built-in overload controls. Examples are: SIP, LDAP, Diameter and H.323 [8]. Moreover, some protocols do not even provide a means to indicate processing overload in response to a service request. Examples are: DNS, Radius, SNMP, SOAP and SAML.

- Session-based and command-response protocols.

- Load-balancing and load forking.

- Proxies and information hiding.

4.5 GOCAP requirements

The following set of top-level requirements are proposed:

1) automatically maximize effective throughput (i.e. admitted service requests/sec) at an overloaded resource subject to keeping response times low and giving priority to follow-on service requests (for session-based protocols); achieve this throughput for the duration of an overload event, and irrespective of the overloaded resource's capacity or of the number of sources of overload (the AMG to CA fan-in issue is an important special case);

2) automatically control processing overload caused by high rates of session release requests;

3) be aware of differing importance levels of messages, and be configurable to reject low importance messages first, and then also reject the next higher importance level as well, and so on;

4) be configurable by the service provider so that, under processing overload, a high proportion of response times at overloaded resources are low enough so as not to cause customers to prematurely abandon service requests;

5) be configurable by the service provider so that, when several inter-acting controls are active at the same time, they converge to an acceptable steady-state when offered loads are constant;

6) inter-work with demand call routing, load balancing and load forking;

7) automatically limit ineffective service requests by detecting specific called names/addresses that are attracting a high reject rate and selectively controlling demand to them;

8) handle translation of called identity (name or address);

9) handle topology-hiding mechanisms;

10) enforce fair allocation of an overloaded processing resource between competing controlled streams of service requests;

11) enforcement of SLAs (to divide the capacity of an overloaded resource between competing streams of service requests according to agreed policies);

12) be able to know which types of service requests may be rejected (by the NGN overload control), and which may not, given the protocol used between two hosts;

13) use the smallest practicable set of standardized, fully-specified, overload control components, in order to minimize the costs of implementation and end-to-end proving;

14) have standardized interfaces (APIs) to the overload control's components for use by the applications running on each host;

15) manual configuration of the overload control's components via a standardized network management interface;

16) (optionally) automatic configuration of the overload control;

17) output network management data output on event occurrence (e.g. control activation/termination) and on demand from network management (e.g. counts of service requests admitted and rejected by the overload control);

18) have adequate security from malicious actions;

19) shall apply within a service provider's NGN, and between different service providers' NGNs;

20) shall apply within an NGN subsytem (e.g. IMS, PSTN/ISDN emulation) and between different NGN subsystems.

5 Overload control design rules

The following is proposed as an initial set of design rules that help ensure that the overload control architecture's design requirements (clause 4.4) should be met.

Internal Overload Control: All resources that can get into processor overload should ideally have a function that can detect processor overload and an admission function that rejects just enough fresh incoming demand to maximize successful completion of admitted sessions subject to not exceeding customer tolerance of long set-up delays.

NOTE: Many telcos' PSTN call processors have such adaptive internal overload controls. They use feedback loops to determine dynamically what part of the stream of initial service requests should be rejected, in order to keep some internal measured quantity (e.g. processor occupancy) roughly constant.

External overload control: Because rejecting fresh calls takes processing effort, effective throughput at an overloaded resource (i.e. its admitted rate) must eventually fall as the load offered to it is increased; and ultimately it will spend all its time rejecting fresh demand. So, to prevent this, it is necessary that controls external to the resource act to reduce the fresh offered load to the level at which its effective throughput is maximized.

Discrimination: Both internal and external controls should:

1) discriminate so that subsequent demand (e.g. session release requests) after the initial request is only rejected if all lower importance demand has been rejected and the demand level is still too high; and

2) discriminate in favour of priority demand (e.g. emergency sessions).

Explicit demand rejection: Preferably, the admission control at an overloaded resource should reject a service request by sending an explicit response rejecting the request, and the backward response should indicate that the request was rejected due to processing overload. The backward response may indicate to an external control what level of restriction to apply.

Failing explicit backward overload indication, which may not be available in many protocols, there are two options. Either:

1) use a pseudo-Admission control at an overloaded resource (see clause 6.1.2); or

2) depend upon the use, at external points of control, of timed-out responses to service requests as a surrogate for explicit backward overload indication.

Both these control configurations make targets vulnerable to misbehaving external sources, and more sensitive to fluctuations in arriving demand, leading to longer response time tails. Also this use of time-outs relies upon response times being predictable and short under non-overload conditions. That may not always be the case. A server's response times may tend to be mainly short but with the occasional long delay when the server has to get non-local information (e.g. DNS lookups, SIP redirection/location proxies).

Closed loop feedback control: The internal and external controls protecting a resource must have a closed-loop feedback structure in which the controlled variable is the rate (per second) at which an overloaded resource rejects service requests due to overload. The goal of the feedback control must be to adjust the level of external restriction so as to cause the controlled variable to converge rapidly to a low value. This choice of controlled variable and goal should ensure that the level of demand arriving at an overloaded resource will automatically adapt to match that resource's capacity, whatever the capacity may be, and however many demand sources are causing the overload (see [2]).

The closed loop structure is illustrated in figure 3.

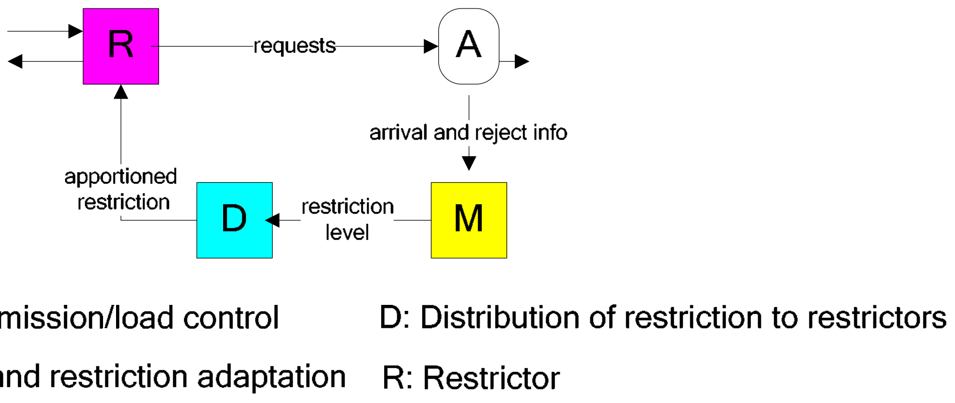
R
requests
A
arrival and reject info
apportioned restriction
D
restriction level
M
A: Internal Admission/load control
D: Distribution of restriction to restrictors
M: Monitoring and restriction adaptation
R: Restrictor

Figure 3: Basic components of an adaptive overload control based on feedback of session rejects

The functional components are as follows:

1) an internal Admission/load control mechanism (A) at the target that controls access to a network resource (including access to the bearer and signalling networks, re-routing under congestion, load sharing, and advanced service features);

2) a reject Monitoring and restriction Mastering function (M) that counts and analyses service request arrivals and rejections by the Admission function (A) and uses that information to adapt the restriction level;

3) a distribution function (D) that apportions and distributes the restriction level calculated by M to each of the restrictors;

4) a restrictor (R) that rejects part of the demand directed to an overloaded resource and is driven by updates from D.

Location of control components: The control components must be distributed between the overloaded resource and external points of control in one of the following two ways:

1) The overloaded resource hosts A, M and D, and so masters and adapts the level of restriction which is applied externally at R. For this to work, the protocol used between the sending and overloaded resources must allow carriage of the restriction level.

2) The overloaded resource hosts just A; and M, D and R run on the sending resource. The feedback from overloaded resource to sending resource is then explicit rejections of requests (carrying an overload indication).

Control loops in series (staged control): It is possible to connect in series several sequences of components R, A, M and D (forming a closed loop control). This is termed staged control, and has the following advantages:

1) lower control loop round-trip times (more stable control);

2) avoidance of problems with translation of called number or address (a control message sent back to an originating node could otherwise refer to an identity that is meaningless to that node);

3) avoidance of too great a degree of "fan-in" from many points of control (R) to a single overloaded resource (faster adaptation of control level).

Restriction method (R): This should be either gapping or leaky bucket (or other method that bounds admitted rates). Gapping and leaky buckets can be implemented in several ways; the following two examples illustrate the concepts:

1) Gapping (Crawford algorithm). Upon admitting a request, start a timer of duration t seconds. Then reject all subsequent requests which arrive at the restrictor before the timer has expired.

2) Leaky bucket. This is a count that continuously decreases at rate r (subject to not falling below 0). When a request arrives to find the count is less than the bucket maximum, then admit the request and increase the count by 1, otherwise the request is rejected and the count is not changed.

Automatic Destination Control (ADC [3])**:** Overload controls should be able to automatically identify and control the load offered to specific called customers or destination network addresses attracting a high failure rate due to congestion at a terminating resource or network resource. This can be done by suitably modifying the Monitoring and Mastering function (M), e.g. by the use of leaky bucket monitors assigned automatically to called identities attracting high failure rates.

SLA (Service Level Agreement) enforcement:. Closed-loop feedback controls protecting a resource can be arranged so that the capacity of that resource is divided between the streams of requests causing the overload according to agreed amounts. For example, requests coming from service "x" can be guaranteed to get at least a proportion "p(x)" of the total capacity. For this to be implementable, each request arriving at the overloaded resource must somehow identify the service it belongs to. For traditional IN services, the identifier is a Service Key; for the service streams in an NGN, there may or may not be any such identifier.

6 Detailed NGN overload control architecture

6.1 Control architecture: General approach

At its top level, the GOCAP architecture provides an optional generic overload control interface (labelled GOCAP in figure 4) acting in parallel to an existing protocol interface (labelled PROTOCOL) being used by the source and target communication applications (labelled as C). This approach does not change the protocol interface, rather the overload control interface is used to affect the behaviour of the communications application at the source. For simplicity the model shows a source / target relationship but it should be noted that on many control interfaces the demand can be sourced by both entities, i.e. in both directions so that either end can be simultaneously acting as both a source and a target. It should also be remembered that there will usually be more than one source that contributes to the load experienced by the target.

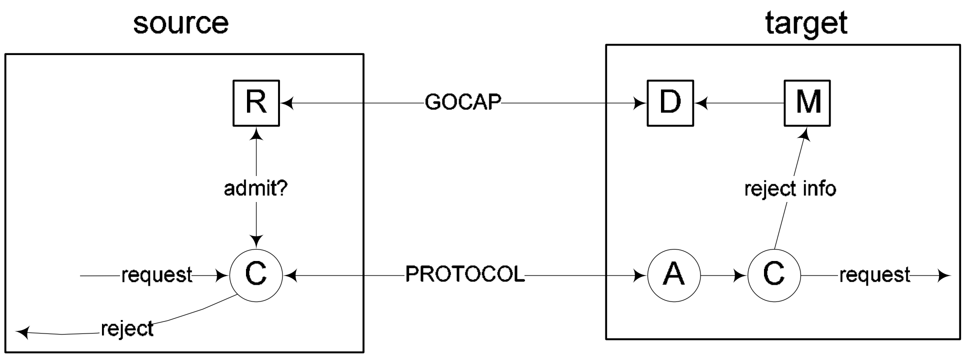

Figure 4: Relationship of Generic Overload Control Interface

6.2 Options for locating components (M, D and R)

Consider an overloaded NGN processing resource (the target) receiving service requests from one or more NGN processing resources (the sources), and consider the protocol in operation between a specific source and the target. Then there are, logically, just 3 cases to consider:

1) the protocol supports rejection (due to overload) of service requests by the target with an indication of overload (e.g. SIP [RFC3261], HTTP [RFC2616], LDAP [RFC3377], COPS [RFC2748], H.323 [8], SOAP, Parlay OSA API);

2) the protocol supports an indication of overload, but not rejection of service requests due to overload (e.g. H.248.11 [7]);

3) the protocol supports neither rejection of service requests due to overload nor indication of overload (e.g. DNS [RFC1035], RADIUS [RFC2865], SAML, H.248 (target = MGC)).

6.2.1 Case 1: protocol supports reject of service requests by the target with an indication of overload

For case 1, the target will have an Admission control. There are two options for locating the overload control's components: option 1 (shown in figure 5) and option 2 (shown in figure 6).

In option 1, M and D are located at the overloaded resource (the target), and R is located at adjacent resources (sources). Service requests arriving at the target's admission control (A) are either admitted or rejected, and handled accordingly by the target's application (C). If a request is rejected, then C copies reject information to M, which mastered the required restriction level. The overload control protocol then sends updates to restriction levels to R using the Distribution function D. At the source, the application (C), asks R whether it can forward a service request to the target.

Note that a key assumption is that the communication applications ('C') are users of the protocol in operation between source and target, but are not part of the protocol. So to add an NGN overload control to an existing NGN processing resource, a vendor only has to change C's code to talk to the NGN overload control APIs (e.g. to ask R for permission to forward a SIP Invite, or report to M that a service request has been rejected due to overload).

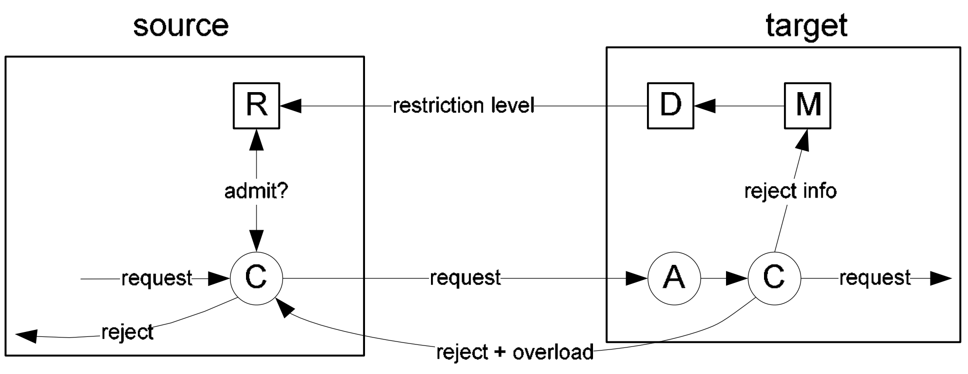

Figure 5: Case 1 - option 1

In option 2, M, D and R are located at the source. The sources application (C) copies rejections with overload indication which it receives from the target to the Monitoring instance (M) at the source, and asks R if it can forward service requests to the target.

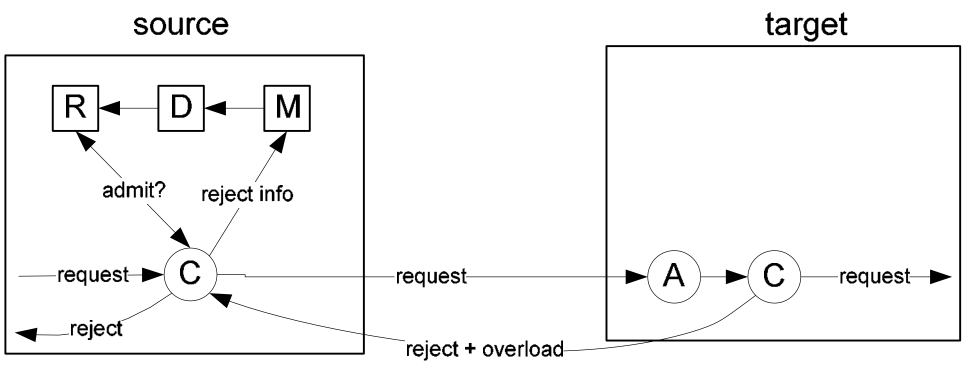

Figure 6: Case 1 - option 2

Option 1 has an important statistical advantage compared to option 2: the target sees the totality of service requests it rejects. That means it can estimate reject rates faster than can option 2, for which each source only sees part of the rejects.

6.2.2 Case 2: protocol supports an indication of overload, but not rejection of service requests

In this case, the target needs a pseudo-Admission function (denoted by "pA" in figure 7) in order to support GOCAP (Figure 7). The pseudo-Admission function marks incoming service requests as "pseudo-rejected" or not, based upon measurements of target loads and delays.

In option 1 (shown in Figure 7), M and D are located at the target, and R at the source. A pseudo-rejected request is not rejected by the target but instead is processed fully, and information about the request is passed to the target's reject Monitor (M), which then acts upon it exactly as if it were a true reject as described for case 1, option 1.

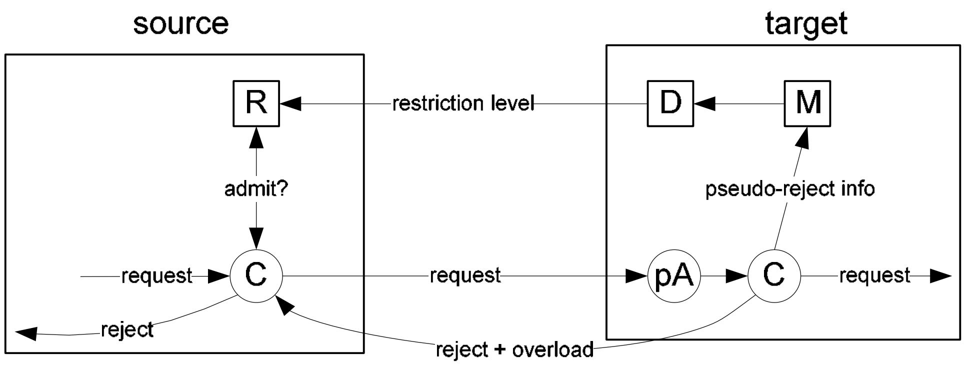

Figure 7: Case 2 - option 1

In option 2 (shown in figure 8), M, D and R are all located at the source. The source application (C) then passes overload indications to its local monitor instance M, and the control aims to throttle back service requests to a rate at which overload indications are received infrequently.

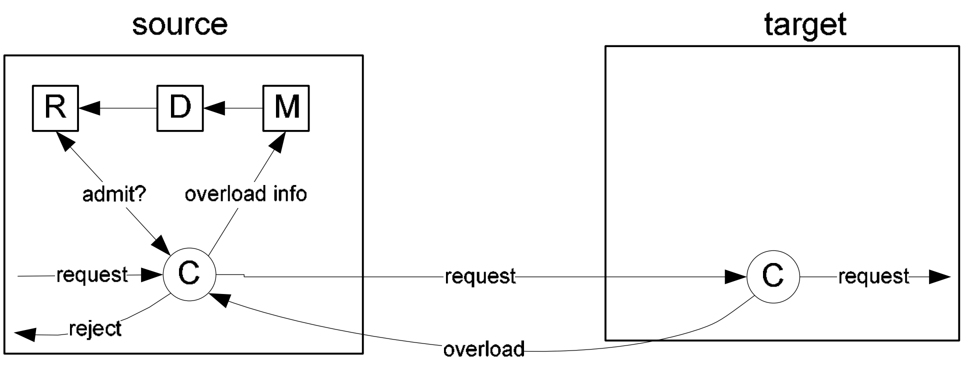

Figure 8: Case 2 - option 2

6.2.3 Case 3: protocol supports neither rejection of service requests nor indication of overload

In this case, the protocol operating between source and target has no way of rejecting a service request due to target overload.

Option 1 (shown in figure 9) has M and D located at the target and R at the source. As with case 2, option 1, the target has a pseudo-Admission function, which marks individual service requests as "pseudo-rejected" or not based upon measurements of the target's internal resource loadings or delays. A pseudo-rejected request is not rejected by the target but instead is processed fully, and information about the request is passed to the target's reject Monitor (M), which then acts upon it exactly as if it were a true reject as described for case 1, option 1.

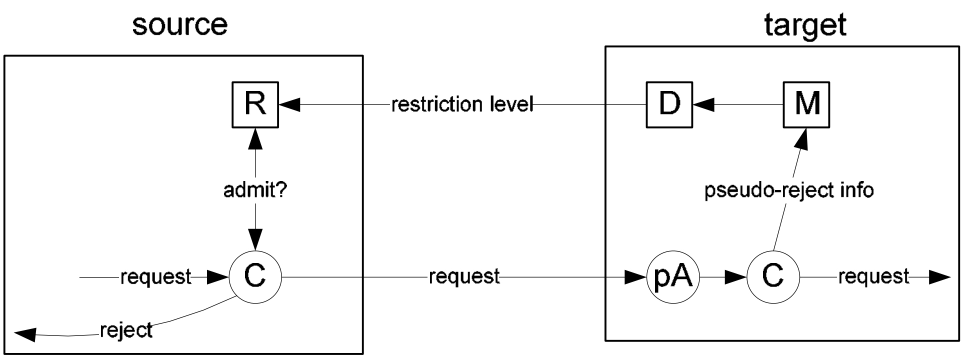

Figure 9: Case 3 - option 1

Figure 10 shows option 2 in which M, D and R are located at the source. The source's application (C) runs a timeout for each service request sent to the target, and passes reject information to M if the timeout expires before a response to the service request is received back. This option either depends upon response time being predictable and short when the target is not overloaded, or depends upon the session control protocol always returning (in a predictable and short time) an indication from the target that the request has been received and is being processed (e.g. SIP 100 Trying).

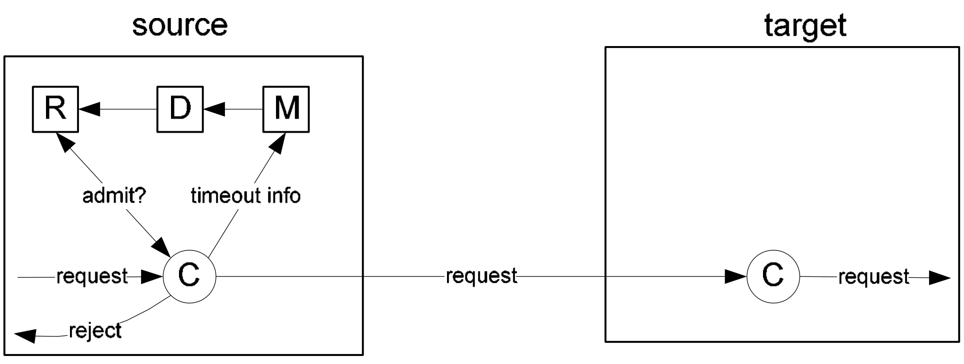

Figure 10: Case 3 - option 2

6.2.4 Preferred locations of components

Based upon the options discussed in the preceding clauses, it is recommended that the GOCAP functional components M, D and R be located as shown in figure 11. That is, the Monitoring and restriction Distribution functions should be located at the target and the Restriction function should be located at the sources.

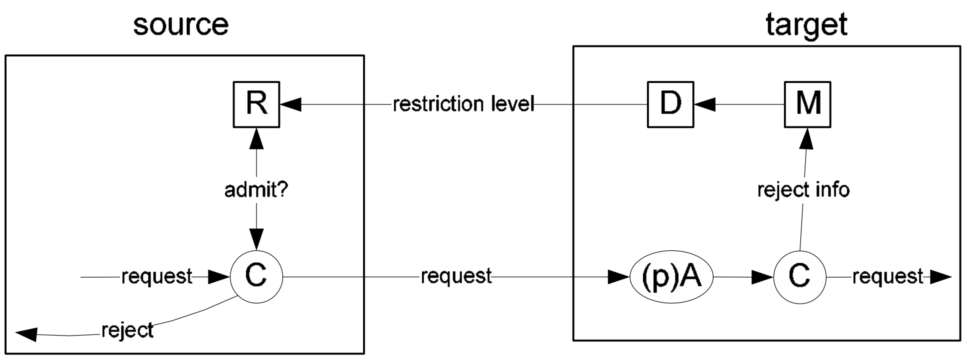

Figure 11: Preferred locations for M, D and R

6.3 Destination load control

The type of overload considered in this clause is one focussed onto a destination address (e.g. SIP User Agent Server) as opposed to node load control described below.

This is illustrated in figure 12, in which a high rate stream of session-related or command/response service requests is focussed on the called address "x" parented on node "y".

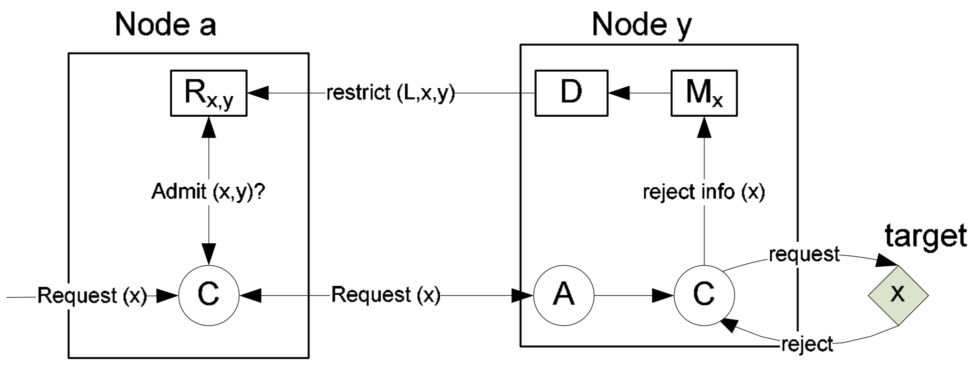

Figure 12: Destination load control

The protocol used between "x" and "y" conveys reject responses from "x" back to the application C in node "y". C notifies the GOCAP instance resident at node "y" of each such reject, passing across the identity "x". GOCAP creates a reject monitor (denoted by M_x) specifically associated with address "x", if one does not currently exist. The Monitor estimates the rate at which "x" rejects requests. If the reject rate exceeds a configurable threshold then control of requests to "x" is started. That is, M starts to send the control variable associated with address "x" to the distribution function, for onward communication to adjacent nodes by GOCAP messages. At adjacent nodes (such as node "a") a restrictor (denoted $R_{x,y}$) is created to throttle service requests sent via next hop node "y" to address "x".

NOTE: It is possible that more than one Restrictor against address "x" could be active at node "a". For example, one restrictor could be throttling service requests forwarded to "x" via "y", and another could be throttling routing requests relating to address "x" sent to an overloaded external routing database. The GOCAP message therefore needs to identify the controlled address "x", the next hop node (y) towards "x" from the point of restriction, and the restriction level (L).

At node "y", M_x periodically updates the restriction level applied against "x" in order to drive the reject rate measured by M_x close to its (configurable) goal reject rate. Each time the restriction level is updated, The distribution function, D, sends the appropriate allocation of that level to each of the adjacent node s. This then ensures that the adjacent nodes receive appropriate shares of the capacity of "x" to handle incoming service requests, if the restrictors employ a method that bounds admitted rates (as do leaky bucket methods, and call gap methods). Control against "x" at node "y" should cease when the current restriction level has adapted down to a (configurable) minimum level, and has stayed at that level for a configurable (termination pending) interval.

6.3.1 Control loops in series

Control of service requests towards an overloaded destination address "x" can be extended beyond the immediate neighbours of the node "y" that hosts "x". This can be done by connecting feedback control loops in series, as illustrated in figure 13.

The advantage is that the streams of requests to address "x" are restricted closer to the points where they enter the network, thus greatly reducing the amount of ineffective demand in the network. In addition, short feedback loops reduce the loop round-trip time improving feedback stability.

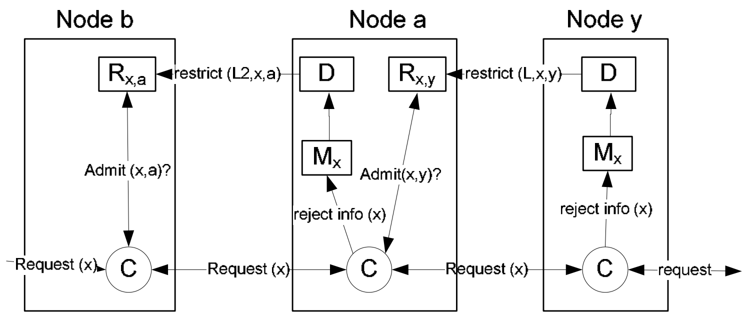

Figure 13: Destination control loops in series

The control loop between nodes "a" and "y" operates as described above. The control loop between nodes "b" and "a" is driven by rejected admission requests at node "a". The application C at node "a" notifies GOCAP each time a service request to address "x" is either rejected by Restrictor $R_{x,y}$, or a (non-GOCAP) response is received from downstream indicating that the request has been rejected due to overload at "x". GOCAP then activates a Monitor M_x at "a". A high enough rate of rejections measured at a by M_x results in restriction levels being distributed periodically to node "b".

6.3.2 Called address translation

Figure 14 shows how two control loops in series can be configured to handle a change to the request destination address at the node where the two loops meet (node "a" in figure 14).

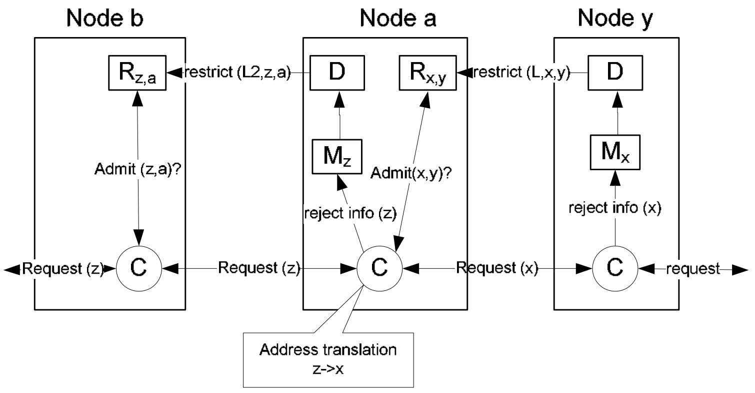

Figure 14: Handling request address translation

Suppose that the application logic "P" at node "a" changes all service requests to address "z" to requests to address "x". When requests to "x" are being throttled at "a", then "C" should therefore notify the local GOCAP instance at node "a" that a service request to address "z" has been rejected, every time the Restrictor $R_{x,y}$ rejects a call to "x". That way a new control loop, acting between nodes "b" and "a" will be created restricting calls at "b" to address "z". Notice that this called address functionality will always work when the mapping function of addresses "z"→"x" is bijective (i.e. each address in set Z mapped to exactly one address in set X and each address in set X has an address in set Z that maps to it). If address wildcarding is allowed, then the mapping of each address range in Z to a address range in X must be bijective. If these properties do not hold, then the reverse mapping at node "a" is non-trivial and is for further study.

6.4 Control of server overload

The type of overload considered in this clause is server overload caused by a general increase in the rate of service requests to a large number of destination addresses (i.e. the request stream is not mainly directed to a single destination user). It is then appropriate to activate controls at all adjacent nodes that are contributing to overload of the server in question; and for those controls to restriction all streams of requests forwarded to the overloaded server.

Figure 15 illustrates how NOCA would be configured to throttle requests sent to the overloading node "y".

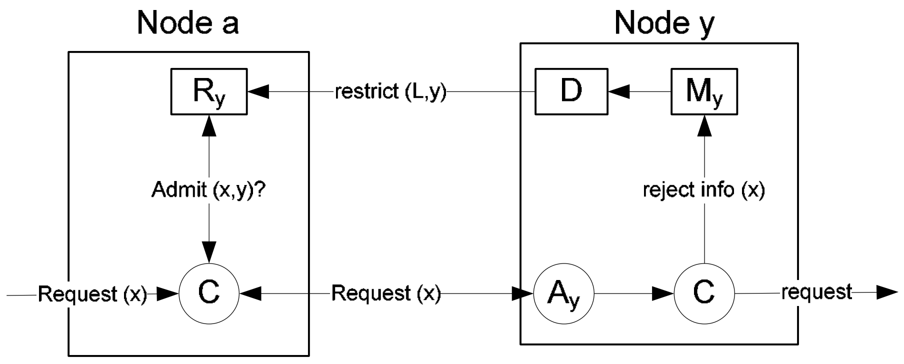

Figure 15: Control of server overload

It is assumed that node "y" has an admission control (denoted by A_y) which decides, based upon internal load monitoring, whether an arriving service request shall be admitted or rejected. If a request is rejected, then an indication of that is passed to the application logic C, so that C can respond appropriately to node "a", and also notify node y's NOCA instance that a request has been rejected due to processing overload of node "y".

The monitoring and mastering function (denoted by M_y), specifically associated with address (i.e. node) "y", measures rejects from "y" and (directly or indirectly) estimates their reject rate over a short interval. If the reject rate exceeds a configurable threshold, M_y then control of requests to "y" is started. That is the (configurable) initial restriction level to be applied against y is returned to an adjacent node (via a GOCAP message) in response to the subsequent receipt of a request from that node, whether or not "y" rejects the request. At adjacent nodes (such as node "a") a restrictor (denoted R_y) is created to throttle service requests sent via next hop node "y".

At node "y", M_y periodically updates the restriction level applied against "y" in order to drive the reject rate measured by M_y close to its (configurable) goal reject rate.

NOTE: The per-node goal reject rate will need to be separately configurable from per destination-user goal reject rate.

Each time the restriction level is updated, that same value is apportioned and distributed by D to each adjacent node in response to a service request received from that node by "y". Then this ensures that the adjacent nodes receive equal shares of the capacity of "y" to handle incoming service requests, if the restrictors employ a method that bounds admitted rates (as do leaky bucket methods, and call gap methods). However, it is possible to divide the capacity of address "y" between the adjacent nodes in set proportions, for example by scaling the restriction level sent to each adjacent node. This might be desirable if the adjacent nodes have different capacities, so that larger nodes got a larger share of "y".

Control protecting node "y" should cease when the current restriction level has adapted down to a (configurable) minimum level, and has stayed at that level for a configurable (termination pending) interval.

6.4.1 Control loops in series

Figure 16 illustrates how NOCA could be configured to extend per-node throttling to protect node "y" beyond the immediate neighbours of "y", assuming next-hop forwarding of service requests. The application logic "P" at "a" notifies the local NOCA instance at "a" that node "a" has rejected a call due to general overload of node "a" every time a service request to node "y" is rejected by the per-node throttle, R_y, protecting node "y". And it does this even though node "a" is not overloaded. This copies the method of connecting of per-called user control loops in series described in clause 6.3.1.

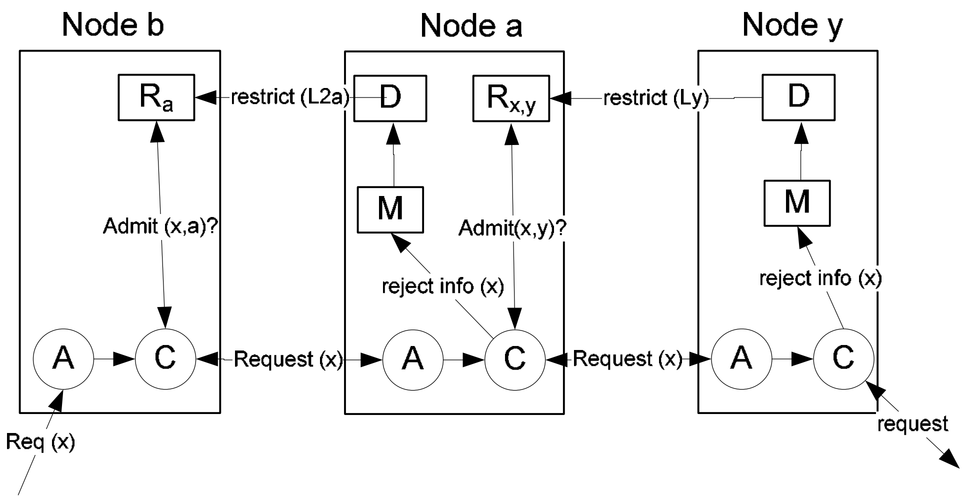

Figure 16: Per-node control loops in series

There is a problem with coupling of control loops for per-node overload control. It occurs when nodes adjacent to node "a" are told to throttle all request streams passing via node "a" even though node "a" is not overloaded, just in order protect node "y". Unlike per-destination throttling, where the controlled destination address ("x") is used to route onwards so that demand to "x" can be throttled, given next-hop routing there seems to be no way that node "b" can know which of its requests will be routed via node "y" unless "y" is adjacent to "b". It is thought, therefore, that propagating per-node control information beyond immediate neighbours is not very useful. It may, however, be worth keeping this as an internal NOCA configuration option.

6.5 Joint control of destination- and server-overload

Figure 17 illustrates how NOCA should be configured to provide joint control of destination- and server-overload. The figure assumes a focussed overload directed to destination user "x" at node "y" that overloaded both "x" and "y". At node "y" service requests rejected by "x" cause activation of a control with Monitoring and Mastering function M_x these applied both at adjacent nodes such as "a" and more distant nodes such as "b" (using control loops in series). Also at node "y", the application admission control (not shown) rejects service requests, and causes activation of a per-node overload control with Monitoring and Mastering function, M_y and restriction R_y applied at adjacent nodes such as "a".

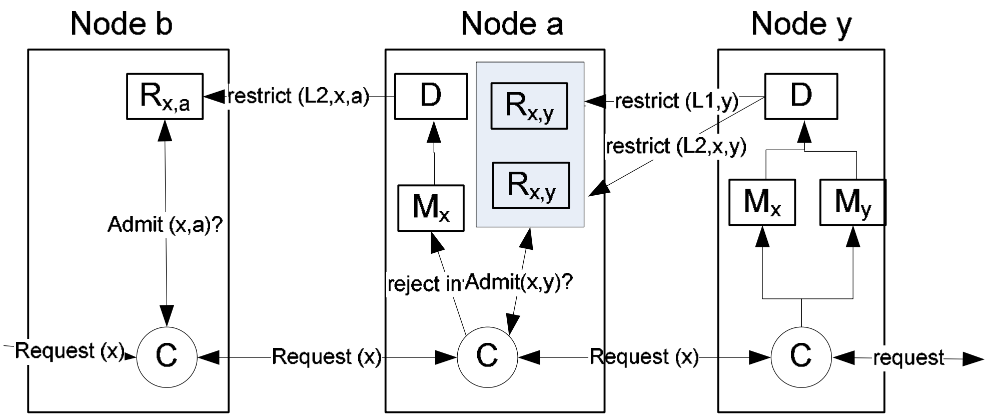

Figure 17: Joint control of destination- and server-overload

At nodes such as "a" at which more than one active restrictor matches a service request, then NOCA should only admit a request to be forwarded if each matching Restrictor admits it.

6.6 AMG to MGC fan-in issue

There is a particular issue with media gateway controllers (MGCs) which may be loaded from a very large number of access media gateways (AMGs). The large number of load sources makes the maintenance of separate load monitors and control update controllers problematic because of the additional processing load placed on the overload target. (In general, there is an issue with any scenario where a resource may become overloaded due to many sources.) It is still desirable to integrate the AMG→MGC overload control with the GOCAP on the MGC as this means that the available processing resource can be allocated between different functions on the MGC in a controlled way. One possible approach for the AMG to MGC overload problem is described below.

A control to limit the load from an AMG is currently being standardized within ETSI ES 283 039-3 [1] and this control enables the MGC to set a percentage restriction to be applied against all non-priority service requests. This can be integrated with a rate limiter based GOCAP running on the target by implementing a pseudo admission control in the communication application on the target that looks at requests from AMGs (or sets of AMGs). In figure 18 we have the communication applications in white circles, GOCAP components in clear rectangles and ES 283 039-3 [1] components in the shaded lozenges. The pseudo admission control, pA, measures the arrival rate, a, from the AMGs and the "reject" rate, ρ, from the GOCAP throttle R_A and uses this rate to update the ES 283 039-3 [1] restriction probability, r. If the current restriction is r_o, then the new restriction, r_n, would be given by an expression similar to $r_n = r_o + \frac{\rho}{a}(1 - r_o)$. The new restriction is passed to the ES 283 039-3 [1] distribution function for distribution to the individual AMGs.

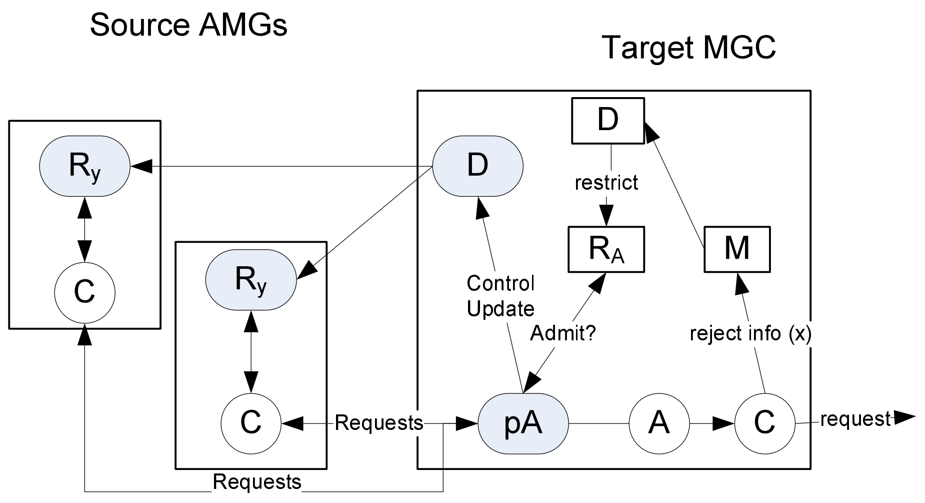

Figure 18: Possible solution to the AMG fan-in issue

6.7 Discriminating between importance levels

It is important that the control system can distinguish between importance levels during an overload episode. One approach could be to use priority levels as described in ITU-Recommendation H.248.11 clause 8.2.5 [7]. Another approach could be the use of leaky buckets in which different request priorities are implemented using a different bucket fill rejection threshold for each priority. Whatever method is used, it implies that part of the request information that crosses the interface between application and NOCA is the request priority. The details of how this could be implemented are for further study.

6.8 Interworking with load balancing and forking

In general the NOCA throttles should be applied to an outgoing request after any load balancing or forking decisions have been made. It may be that, in the event of load balancing, the rejected request would be offered to another server from the balanced set - but that would depend on the application. In the case of load forking, a rejection of one (or more) of the generated requests by a NOCA throttle should not imply that all the forked requests should be rejected. It should be noted that when the load balancing is performed by a separate entity, rather that the request source, then the comments regarding information hiding below applies. In this case the information that is hidden is the actual identity of the server that will satisfy the request.

6.9 Interworking with information hiding proxies

In some applications, proxies are used to hide information between different functions in the NGN network. This may be for reasons of security, or it may be to hide complex implementation issues from other functions. In these circumstances, the use of the NOCA implies additional functionality in the proxy. Consider figure 19, which shows two request sources able to load three potential overload targets. In this scenario, if Target X is overloaded, then it will use the GOCAP protocol to instantiate and control throttles on sources A and B, the distribution function controlling the apportionment of processing resource at X between A and B.

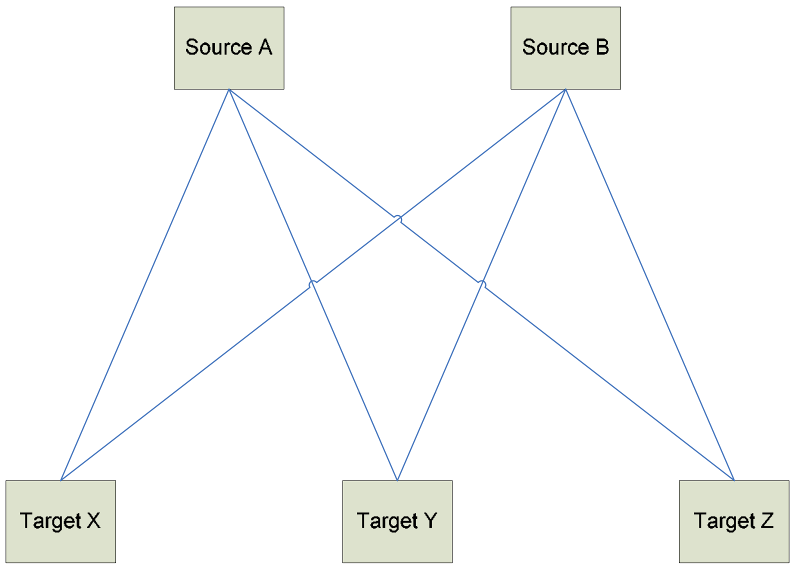

Figure 19: A meshed implementation of sources and targets

In figure 20 we see that a proxy has been added to hide the complexity of the layer below. Now all requests destined for X, Y or Z are sent by A and B to the proxy, P. (For an example of this in the NGN architecture, consider RACS. In this case the sources A and B would correspond to the AF instances, the targets X, Y and Z would correspond to A-RACF or BGF instances and the proxy, P, would correspond to the SPDF.) When the target X detects that it is overload what should it do? It can no longer request throttling at A and B as they are hidden by the proxy. Nor could X request the proxy to forward restriction requests to A and B, as A and B do not know which requests are targeted at X (that matching is done by P). In some special cases, it could be possible for the proxy to use address translation to give request address ranges that have significance at A and B which could be used to throttle requests to X, but often that will not be the case. (See the discussion on address translation above, a bijective mapping of the address ranges associated with X, Y and Z to address ranges used by A and B are possible, though the use of the proxy for complexity hiding would tend to suggest it is unlikely.)

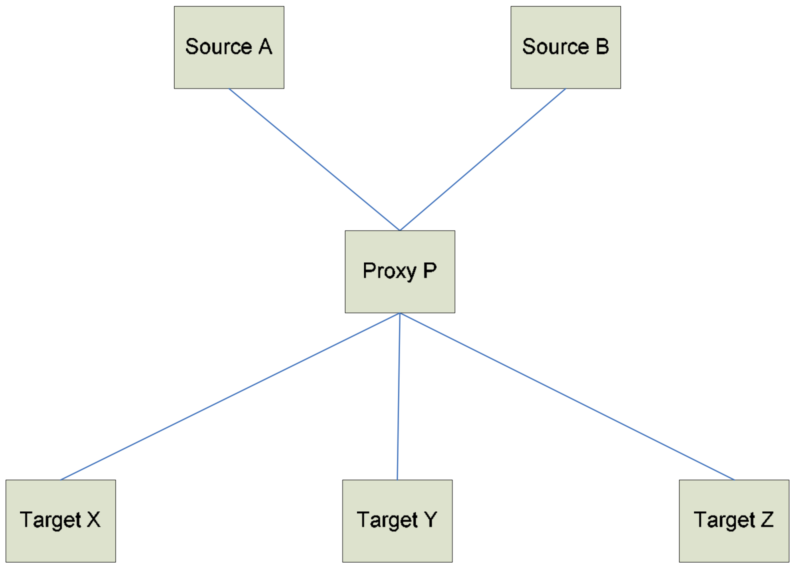

Figure 20:The same scenario using an information hiding proxy

In general terms, X can only invoke throttles at the proxy, which forces the proxy to have sufficient functionality to support GOCAP. The rejection at P of demand destined for X may cause P to become overloaded, and thus P would request throttling of requests to P at A and B. This is almost what would happen in the case without the proxy, except that the throttles deployed at A and B do *not* discriminate between requests destined to X (which is overloaded) and requests destined to Y and Z (which are not). This means that an overload at X can result in the rejection of requests destined to Y and Z even when Y and Z have sufficient resources to handle the requests.

6.10 Multiple controls at an overloaded resource

Figure 21 illustrates how the overload control architecture can be used to inter-work with the following sources:

1) sources equipped with NOCA components;
2) sources equipped with an overload control that is driven by reject messages indicating that the target is overloaded (illustrated by ISUP ACC enabled source in the figure);
3) sources with no overload control; and
4) sources with an overload control driven by restriction messages as part of the protocol used to communicate with target (illustrated by the use of the draft ES 283 039-3 [1] at the AMG in the figure).

Note that in figure 21 the NOCA architecture's components are indicated by boxes, and the non-NOCA functions are indicated by ovals.

The target has a single monitoring and mastering function (M) which is monitoring the total arrival and reject rates and mastering the restriction level. The restriction level is passed to the distribution function (D) which apportions the restriction among the different NOCA managed restrictions using operator defined weights.

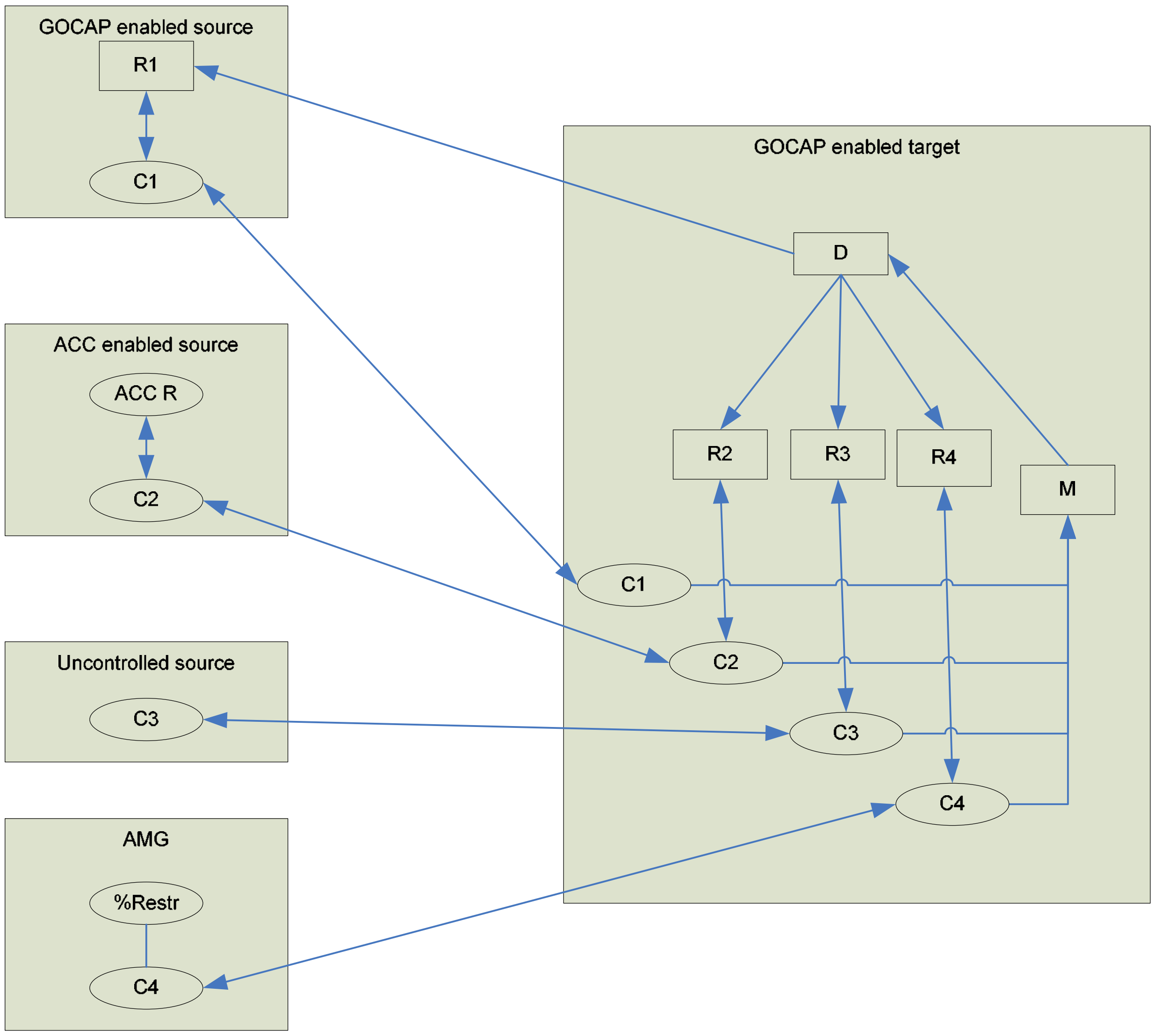

Figure 21: Working with existing or no controls - target is NOCA equipped

For the NOCA enabled source, the feedback loop is as described above, but the target manages three restrictions locally, one for each of the other sources. Assuming the restrictions are leaky buckets, then the weights used by the distribution function could be chosen to ensure that each of the 4 sources gets an equal share of the overloaded resource. Alternatively the weights could be chosen to share the overload resource in proportion to the normal load generated by each of the sources.

When a service request arrives at the target, its Communication Application(s) should check if there is a matching local restriction; and if there is, ask whether the request is to be admitted or not. Note that, if the target has a local Admission Control (not shown in figure 21), then it is assumed that each arriving request will also be tagged to indicate whether it has been rejected by the local Admission Control.

If an arriving service request is rejected at the target, either by the target's Admission Control or by an active Local NOCA throttle, then the logic of the protocol used between the request's source and the target should be followed. In the examples shown in figure 21 this means that a rejected request from source 2 (running ISUP) should cause an ISUP release message to be returned; and the release should indicate target overload (ACL set to 1 or 2) and reason cause (42). A rejected request from source 3 (supporting no overload control) should be returned whatever indication of target overload the protocol provides. A rejected request (in the example, an Off-hook notification) from source 4 should follow the logic of H.248.1 [9] and the H.248.1 [9] extension package that embodies ES 283 039-3 [1]. In particular, the ES 283 039-3 [1] logic running on the target should be applied.

This method scales to any number of external sources.

This method of working with existing non-NOCA overload controls, and uncontrolled sources, has three advantages as follows:

- existing overload controls do not need to be changed either at the target or at external sources (although some change to the Communication Application is required at the target to check against local NOCA restrictions);
- the share of the target's capacity each external source, across different protocols, is managed via NOCA by configuration of the restriction distribution weights;
- the rate of admitted service requests from uncontrolled sources is configurable.

6.11 Specification of standardized control components

6.11.1 Mapping of the M, D & R functions

The overload control architecture has 3 main components, indicated by the boxes in figure 22. Their alignment with the M, D and R functions is as follows:

- Monitoring and Mastering Function (M):
    - This function is realized with two components, the Adaptation Manager and one or more Control Adaptors. These two components are responsible for adapting the control variable(s) by monitoring the load information provided by the application (reject rates or cpu occupancy for example). Each active control at the host is managed by a single Control Adaptor, with the Adaptation Manager responsible for the instantiation and deletion of each adaptor.
- Distribution Function (D)
    - This function is realized by the Control Distribution component. This component receives control updates from the control adaptors and distributes apportioned restriction levels to the throttle controllers at appropriate nodes.
- Restriction Function (R)
    - This function is realized with two components, the Throttle Manager and zero or more Local Throttles. The Throttle manager receives restriction updates from the Control Distribution component (either local or remote) and responds to application queries for request forwarding (by granting or refusing those requests). The throttle manager is responsible for the instantiation and deletion of each throttle.

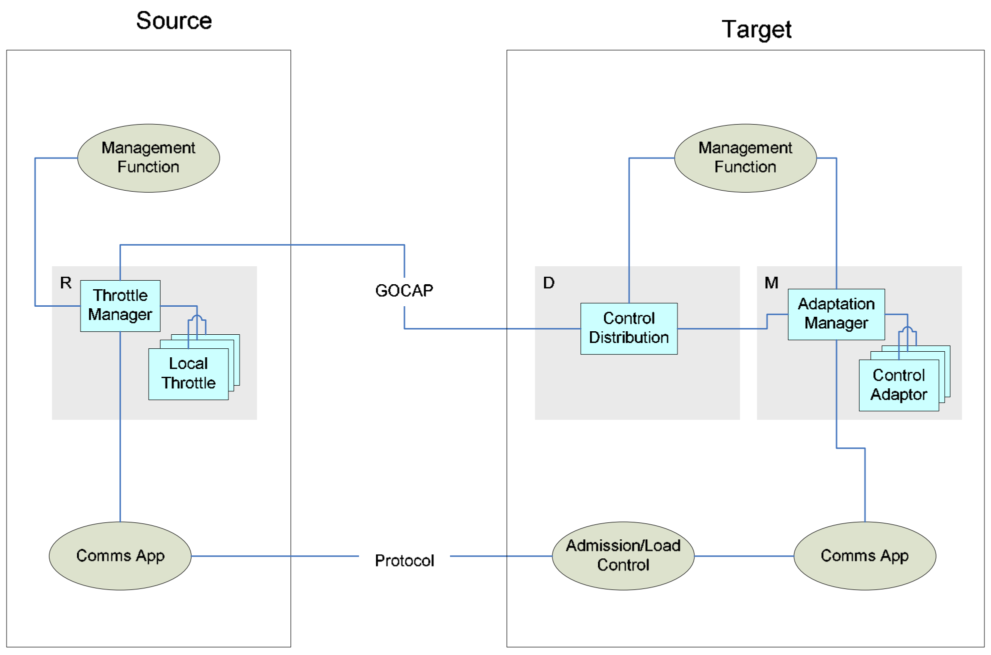

Figure 22: Control components (boxes) and non-NOCA system functions (ovals)

The components and functions shown in Figure 22 should be thought of as being located above the top level of the protocol stacks running at target and source. As an example, the two Comms Application instances running on the source and target may use ISUP to communicate with each other in establishing and clearing-down voice calls. Beneath ISUP, the protocol stack (in descending order) could be SIP-I/TCP/IP, SIP-I/SCTP/IP or SIP-I/UDP/IP.

Various non-NOCA system functions need to inter-act with the control components to form a working control. They are indicated by the oval boxes in Figure 22, and are as follows:

- Management Function:
  - This provides a management interface for NOCA components for configuration, status enquiries, control statistics etc.
- Communications Application:
  - This runs the logic of a service application (e.g. VoIP session establishment, routing, charging etc). There may be several different communication applications running on a server if the server hosts several different services.
- Load Monitor/Admission Control:
  - This provides time series of arrival rates, reject rates, processor occupancy etc. In the case of admission control, it is assumed that the function is already realized as part of the internal overload management for the communication application running on the server. (This is a server load-shedding mechanism that ensures that admitted requests are handled with adequately short response times.) In cases where admission control is not appropriate, a load monitor is required to provide the NOCA control adapters with equivalent data in the form of request arrival rates and pseudo reject rates.

Of course, the splitting of functionality between admission/load control (A) and communication application (C) is somewhat arbitrary - in fact the admission/load control could well be considered as part of the application. It is certainly application aware (so that it knows which messages to apply admission control to). For the description here, it is assumed that load/reject information is relayed to NOCA controls by the communication application.

Conceptually, the NOCA- and system-components communicate with each other by sending requests and returning responses.

A server might be a target in one overload event and a source in another. In that case, it would need the full set of NOCA components configured and running (in a wholly NOCA network). A server could also be a target and a source at the same time if it were both overloaded itself and the source of overload elsewhere. This is shown in figure 23. Note that the application checks the Throttle Manager for admission throttles that apply to requests as they arrive at the node as well as checking before sending a request on to another node. Although remote throttles are preferred, local admission throttles may be used when interfacing with other overload controls as described above.

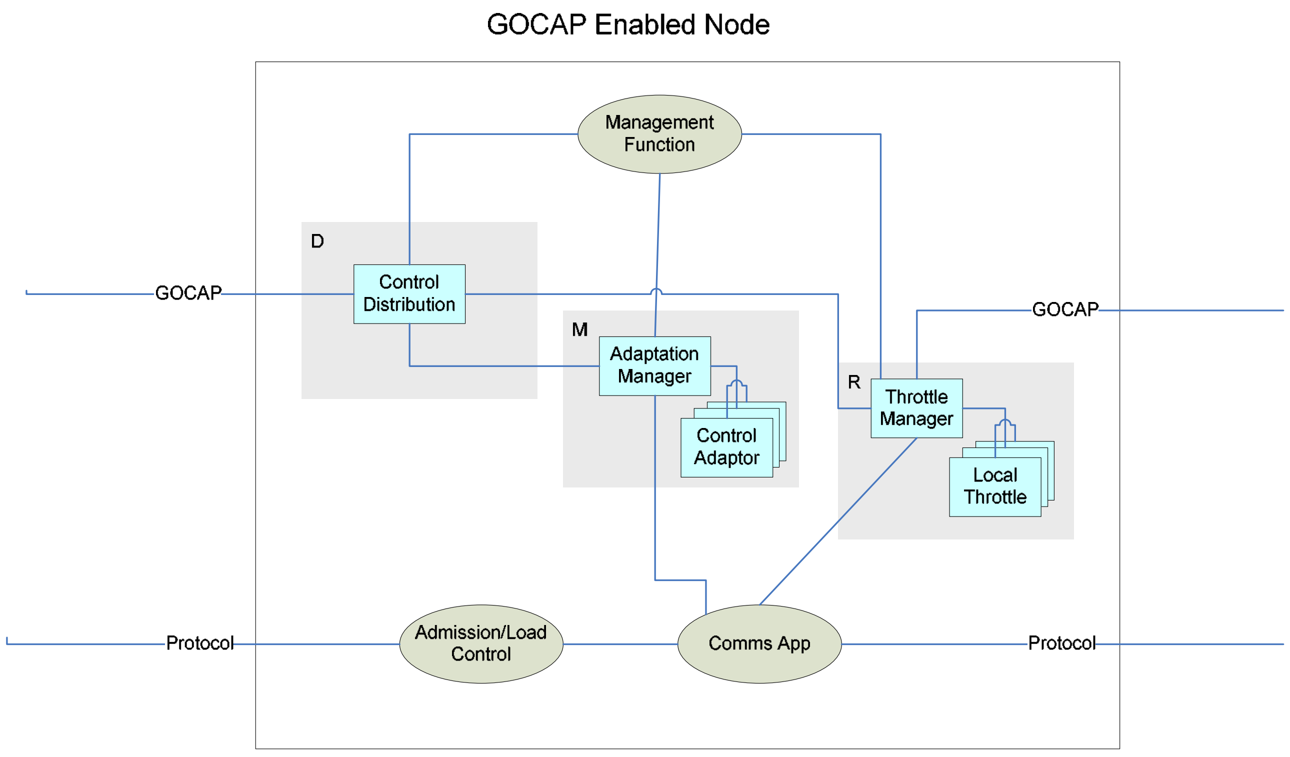

Figure 23: Structure of a NOCA enabled server

6.11.2 Detailed component description

6.11.2.1 Control Distribution component

A Control Distribution component is located at each NOCA target server.

It is created and configured when the NOCA subsystem is started. It receives control variable updates from the Adaptation Controller. The information passed to it in a control variable update is the following (some of which may be pointers to the actual data):

- a list of all (source node, protocol) pairs the control variable should be sent to;
- the identity of the target server;
- the value and nature (maximum admission rate or admission percentage) of the (unscaled) control variable; and
- the ControlDuration parameter value.

It then refreshes its list of (node, protocol) pairs for which a control update awaits dispatch as follows:

1) adds to the list any control updates for (node, protocol) pairs not in the list;
2) over-writes the update data for matching (node, protocol) pairs in the list;
3) scales the control variable value given the weight that applies to a (node, protocol) pair.

The scaling data is configured via the management system. The ideal scaling values may be affected by topology and capacity changes. Sub-optimal scaling factors may affect control performance, especially transient behaviour when the control starts. When choosing scaling values, due consideration needs to be taken of the of the impact of topology and capacity changes on the control performance.

When informed that a service request from a specific (node, protocol) pair has arrived at the target, the Control Distribution component extracts the relevant data for that (node, protocol) pair from its list. If the data is a control update, then it assembles the following data:

- the (source node, protocol) pair the control update should be sent to;
- the identity of the target server;
- the value and nature (maximum admission rate or admission percentage) of the (scaled) control variable; and
- the ControlDuration parameter value.

If the data is a TerminateRestriction command, then it assembles the following data:

- the (source node, protocol) pair a TerminateRestriction message should be sent to;
- the identity of the target server.

If the node identity = target node identity, then the update applies to a Throttle at the target, and the data is passed to the local Throttle Controller. Otherwise, the data is passed to the Throttle Controller at the remote source.

When informed by the target Adaptation Controller that control against a (node, protocol) pair has ended, then it adds the following data to the list of control updates to be sent out:

1) the (source node, protocol) pair a TerminateRestriction message should be sent to;
2) the identity of the target server.

6.11.2.2 Adaptation Manager

An Adaptation Manager is located at each target server.

The Adaptation Manager is created and configured when the overload control is armed.

When initialized, the Adaptation Manager creates a special Control Adaptor to monitor load state of the host and master the NOCA node overload control. This special Control Adapter may be driven by:

- request arrival rate and local admission control reject rate;
- system load state information; or
- system capacity (either static or dynamically derived).

The Adaptation Manager may create, on demand, additional Control Adapters, to support destination load control for example, that are driven by arrival and reject rate data provided by the application.

The Adaptation Manager sets what kind of control variable each Control Adaptor is to use - the two options are: admission percentage and leak rate (the latter applies to leaky buckets and to gaps). And it tells it the other relevant configuration data - including the following:

- initial value of the control variable to send out when feedback control starts;
- Control Duration time interval;
- maximum and minimum values of the control variable;
- goal request arrival rate from the set of (source node, protocol) pairs;
- the lower threshold on the arrival rate;
- the TerminationPending time interval.

6.11.2.3 Control Adaptor

Each Control Adaptor is periodically passed data on the goal arrival rate and the actual arrival rate (or other load information), and acts upon them as specified in clause 6.12.3 to initiate closed-loop control, update its control variable, and terminate closed-loop control.

6.11.2.4 Throttle Manager

The Throttle Manager is created and configured when the overload control is armed.

It manages 0 or more active Throttles.

The Throttle Manager starts and stops Throttles when told to by Control Distribution components (either local or remote). It is informed of the nature of the Throttle, and the specific throttle "address" (source node, protocol, target node, destination) it applies to. (Some fields may be wild carded - e.g. for node load control, the destination field would be a wildcard.) When the Throttle Manager is told to start a Throttle it does the following:

1) adds the throttle "address" to a list of active throttles;
2) instantiates and initializes the Throttle - based on knowing the nature of the Throttle (i.e. percent admission state = initial percentage, leaky bucket state = bucket fill, max fill, and leak rate, or gap state = remaining gap interval).

When the Throttle manager is told to stop a Throttle by a Control Distribution component, it will be given the throttle "address" that it applies to. The Throttle Manager then frees any resources used by the Throttle, and removes the Throttle from its list of active Throttles.

When the Throttle Manager receives a control update from the (local or remote) Control Distribution component, it then commands the Throttle to apply the new value of the control variable (optionally randomizing first use of it to prevent synchronization of Throttles).

When the Throttle Manager receives a query from a local Communication Application whether a service request via a specific protocol to a specific target node can be admitted, it looks for a matching active throttle in its list of active Throttles. If there is no match, then the query is granted, and the Throttle Manager responds to the Communication Application. If there is a match with an active throttle then the Throttle Manager asks the Throttle if the request can be admitted. The Throttle then grants the query or not, and the Throttle Manager replies to the Communication Application. If multiple throttles match the query, then the request must be accepted by all the throttles in order to be accepted. This is coordinated by the throttle manager the throttles may need to be updated with the result of the admission request.

The management function may also request the creation and destruction of throttles by the throttle manager, allowing the network operator to use the NOCA infrastructure for manually applied restrictions as required.

6.11.2.5 Throttle

Throttles are instantiated and destroyed by the Throttle Manager. They are responsible for the actual request accept/reject decisions.

6.11.3 Control of the total load offered to a target node

6.11.3.1 Initiation of closed-loop control

The Adaptation Manager, associated with a resource, will have assigned a Control Adaptor to monitor the following quantities:

- periodically-updated estimates of the total arrival rate of service requests Y(C) at the resource; and
- periodically-updated values for total arrival rate goal Y_G for service requests to that resource.

When the total request arrival rate exceeds its goal rate, then the Control Adaptor changes state to Closed-Loop control. It will have been assigned an initial value for its control variable, and it should send that value out as a control update to the Control Distribution component.

The ControlAdaptor does this by sending a Restrict message to the ControlDistribution component. The message should contain the following data:

- a list of all (source node, protocol) pairs being controlled by the Control;
- the identity of the target server;
- the value and nature (maximum admission rate or admission percentage) of the control variable; and
- the ControlDuration parameter value.

The ControlDuration parameter ensures that a Throttle will cease throttling after a time period equal to the ControlDuration value has elapsed, unless it receives further Restriction messages before then. It is a failsafe mechanism to ensure that a source control will eventually end if the GOCAP association with the target server has failed.

6.11.3.2 Adaptation of control variable in the closed-Loop state

While in the Closed-Loop state the ControlAdaptor receives the following data:

- periodically-updated estimates of the total arrival rate of service requests Y(C) at an overloaded resource; and
- periodically updated values for total arrival rate goal Y_G for service requests to that resource;

and uses it to update its control variable C as follows.

Suppose that Y(C) is a monotonically increasing function of the control variable C, and that the slope of Y(C) is non-increasing everywhere. This is the case for the following throttle types:

Leaky bucket, with C = maximum bucket leak rate;

Crawford Gap, with C = 1/gap interval;

Percentage thinning, with C = admitted percentage.

Then (see figure 24) given the current value C_i of the control variable and the corresponding total request arrival rate $Y(C_i)$, draw a straight line from the point $(C_i, Y(C_i))$ to the coordinate origin, and find the value of C at which this line intersects the horizontal line $Y = Y_G$. Take this as the new value of the control variable: $C_{i+1} = C_i Y_G / Y(C_i)$. .

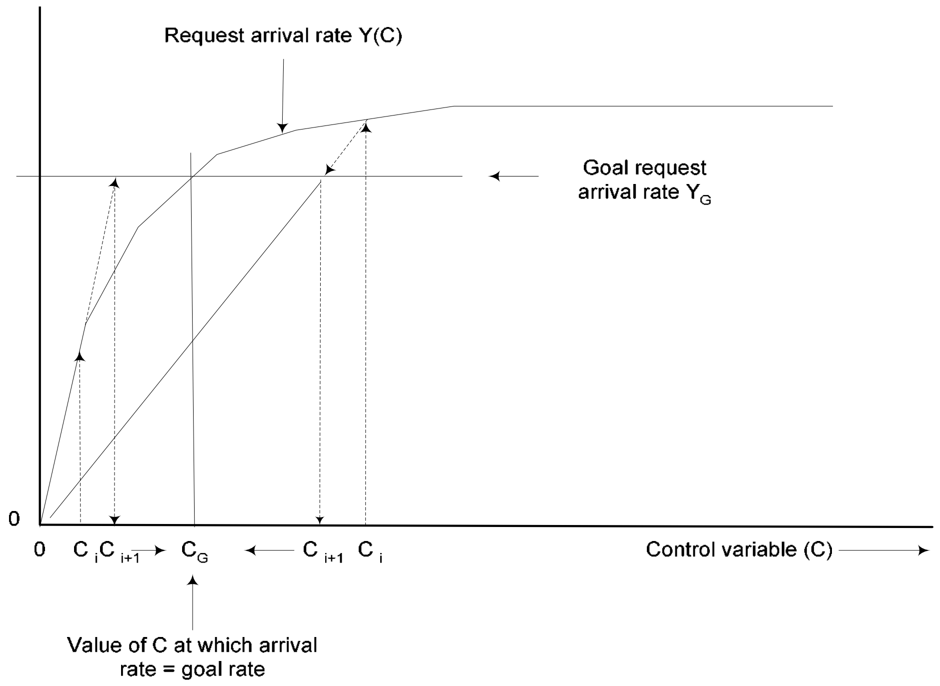
Request arrival rate Y(C)
Goal request arrival rate Y_G
0
0
C_i
C_{i+1}
C_G
C_{i+1}
C_i
Control variable (C)
Value of C at which arrival rate = goal rate

Figure 24: Adaptation of control variable

Then, given the two assumptions about Y(C), it follows that if $C_G < C_i$, then $C_G <= C_{i+1} < C_i$; and if $C_G > C_i$, then $C_G >= C_{i+1} > C_i$. That is, the sequence of successive values of the control variable either decreases monotonically and is bounded below by C_G, or increases monotonically and is bounded above by C_G. In either case the sequence of C values must therefore converge (theorem in real analysis); and the limit is C_G.

The value of C_i should be changed to comply with configurable minimum and maximum values, if its value is either less than the minimum or greater than the maximum allowed.

Deterministic modelling suggests that, in a network where several servers are overloaded and each applies this adaptation method to its (neighbouring) sources, the entire set of interacting nearest-neighbour controls converges - typically in 10 to 20 iterations.

Note that if C = percentage admitted, then Y(C) is an increasing linear function of C, and adaptation takes just one step. However, a percentage-thinning throttle does not bound its admitted rate (unlike the Crawford gap and leaky bucket); and, for that reason, the rate restriction methods are preferred. Synchronization of gaps or leaky buckets can occur when applied at multiple sources. This problem is simply solved by appropriately randomizing the first use of a newly received gap rate or leaky bucket rate.

Figure 25 shows that the adaptation algorithm will cause the control variable to increase without limit when the controlled sources, between them, are originating service requests destined for the target server, at a rate less than the target server's goal arrival rate Y_G . It would be dangerous to allow this to occur because the sources could suddenly increase their offered rates to the target.

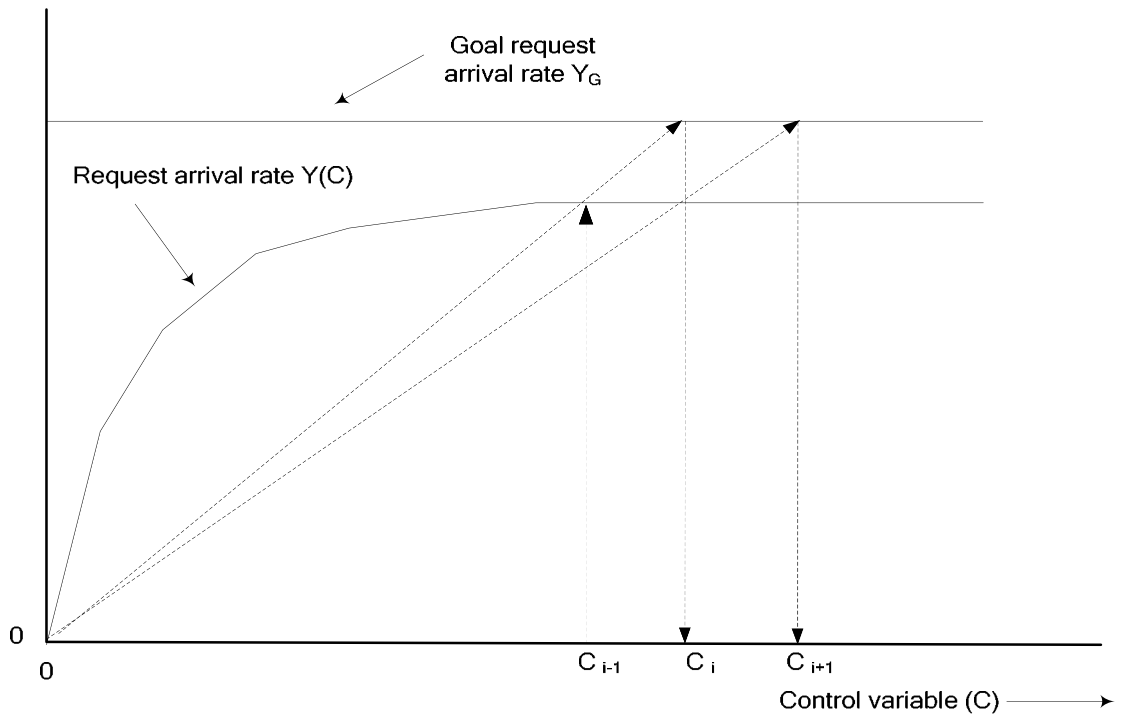

Figure 25: Adaptation when sources originate requests at a rate < goal arrival rate

Therefore, in the Closed Loop state, if the current arrival rate $Y(C_i)$ is not significantly greater than the previous arrival rate $Y(C_{i-1})$, that is if $Y(C_i) - Y(C_{i-1}) <$ MinArrivalRateChange , where the configurable parameter MinArrivalRateChange is small and positive, then the next value of the control variable, C_{i+1} , should revert to C_{i-1} .

When the ControlAdaptor has a control update it sends a Restrict message to the ControlDistribution function. The message should contain the following data:

1) a list of all (source node, protocol) pairs being controlled by the ControlAdaptor;
2) the identity of the target server;
3) the value and nature (maximum admission rate or admission percentage) of the control variable; and
4) the ControlDuration parameter value.

6.11.3.3 Termination of closed-loop control

The ControlAdaptor enters a TerminationPending state when the measured total request arrival rate falls below a lower threshold on the total arrival rate. It starts a TerminationPending timer, sends a TerminateRestriction message to the Control Distribution component. The TerminateRestriction message should include the following data:

1) a list of all (source node, protocol) pairs being controlled by the Control; and
2) the identity of the target server.

If the Termination Pending timer expires while the Control is in the TerminationPending state then the ControlAdaptor reverts to monitoring arrival and goal rates.

If the measured total request arrival rate exceeds the goal rate while the ControlAdaptor is in the TerminationPending state, then control reverts to Closed-Loop control.

6.11.3.4 Allocation of target capacity between controlled sources, fairness and SLAs

If the Throttles are leaky buckets, then a simple and flexible mechanism is available to apportion the target server's capacity among the controlled sources. The mechanism ensures that a source with high enough offered load will get at least a defined proportion of the target's goal arrival rate.

Suppose that there are "n" sources all running leaky bucket throttles. The Target ControlAdaptor's control variable "C" is then a rate, which can be scaled by the ControlDistribution component before being sent out; so that, for example, source i is sent the leaky bucket leak rate s_i C. Impose the conditions that each s_i lie between 0 and 1, and together they sum to 1. Then, when used in the apportionment method to be described, s_i is the guaranteed minimum proportion of the target server's goal arrival rate that source i gets, provided it originates requests at a high enough rate.

Denote by x_i the rate at which source i offers requests to its throttle. Then the admission rate of the throttle as a function of its leak rate s_i C is as follows. When the bucket leak rate exceeds x_i then the whole stream originating at source i is admitted at rate x_i; and when the bucket leak rate is less than x_i then the admitted rate equals the leak rate. (Strictly speaking, this is true only if the leaky bucket has infinite maximum fill. However, it can be closely approximated by setting a leaky bucket's maximum fill to about 10).

So the admitted rate at source i as a function of the control variable C equals s_i C if $C < x_i / s_i$, and equals x_i if $C > x_i / s_i$ (see figure 26).

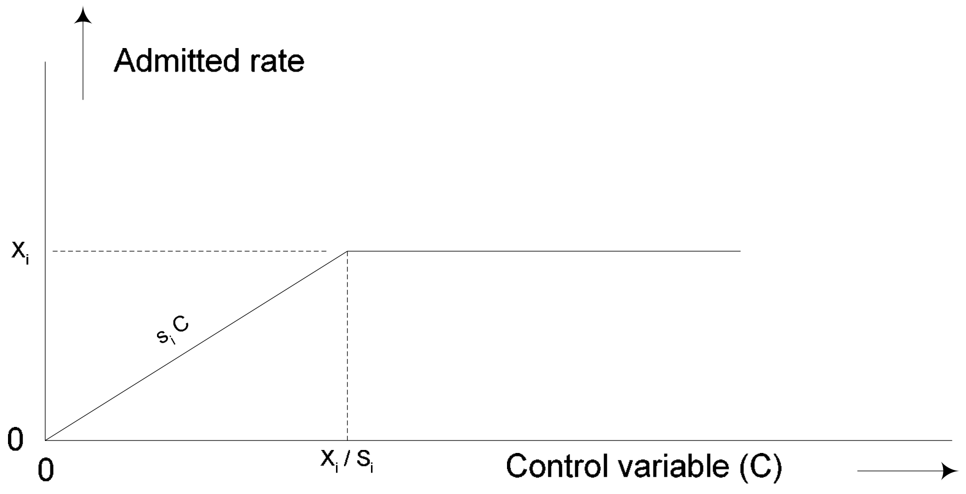

Figure 26: Idealized leaky bucket admission rate function

Summing the "n" source node admission functions, gives the total arrival rate Y(C) at the target node.

Number the sources so that $x_1/s_1 < x_2/s_2 < \ldots < x_n/s_n$. Then:

- For $C < x_1/s_1$, $Y(C) = \sum_1^n s_i C$, and Y(C) has slope $\sum_1^n s_i$. At $C = x_1/s_1$, source 1 reaches its maximum admitted rate of x_1, after which its slope is zero. If the value of the goal arrival rate Y_G gives $C < x_1/s_1$, then $Y_G = \sum_1^n s_i C$, and source i gets the $s_i C = Y_G \frac{s_i}{\sum_1^n s_i}$ for all i. That is, the goal rate is divided between the sources in proportion to their weights.
- For $x_1/s_1 < C < x_2/s_2$, $Y(C) = x_1 + \sum_2^n s_i C$, and the slope of Y(C) is reduced to the value $\sum_2^n s_i$ (source 1 having slope zero). At $C = x_2/s_2$, source 2 reaches its maximum admitted rate of x_2. If the value of the goal arrival rate Y_G gives $x_1/s_1 < C < x_2/s_2$, then $Y_G = x_1 + \sum_2^n s_i C$. The goal rate is divided between the sources so that source 1 gets x_1 and source i gets $s_i C = (Y_G - x_1)\frac{s_i}{\sum_2^n s_i}$. That is, source 1 gets all its stream, and the remaining goal rate capacity $(Y_G - x_1)$ is divided among the remaining sources in proportion to their weights.
- This pattern continues, with the slope reducing by s_i as C passes through x_i/s_i, until the last source attains its maximum admitted rate at $C = x_n/s_n$, after which the slope is zero. That is, Y(C) remains constant at the value $\sum_1^n x_i$.

The graph of Y(C) is illustrated in figure 27.

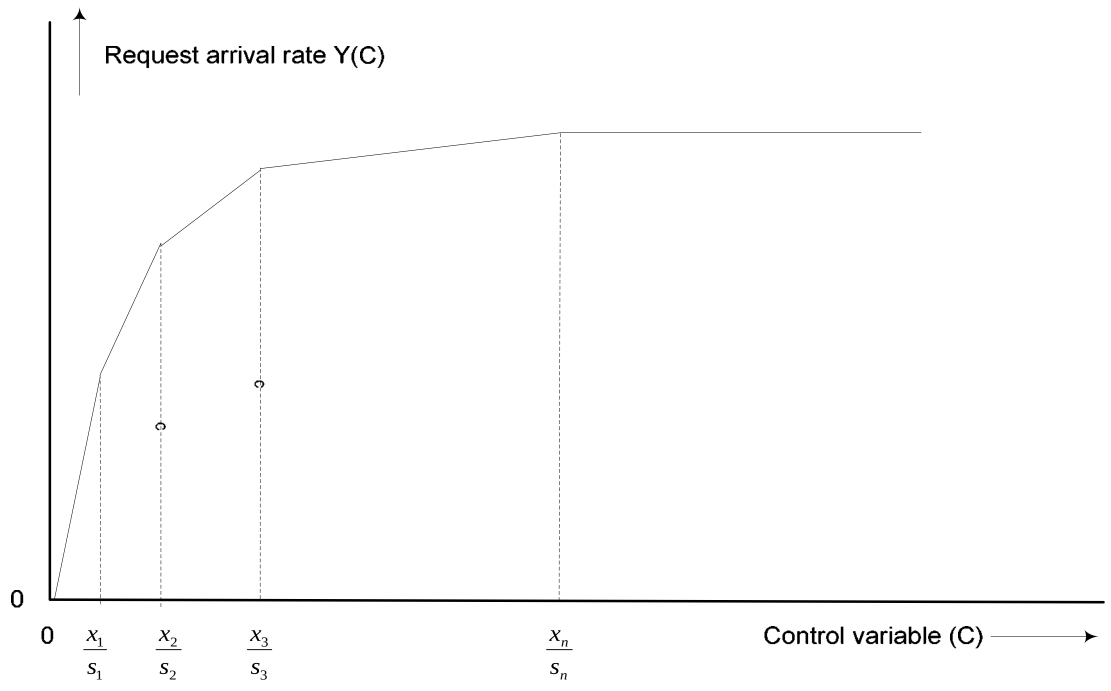

Figure 27: Total arrival rate function Y(C)

A nice feature of this capacity apportionment method is that (generally) sources originating little load will get all of it forwarded to the target, leaving the remaining goal capacity to be divided between the high load sources in proportion to *their* relative weights. This method of apportioning the goal arrival rate is not the same as reserving a set proportion of it for each source.

The apportionment method could be used to divide the goal arrival rate between competing services if the service type of a request is known somehow (for example, deduced from its payload). The method therefore might provide a simple way to enforce an important aspect of service level agreements with different Service Providers.

6.12 Specification of APIs for control components

The interface between the NOCA and other system component (communication application, management system, etc. will be specified. Figure 28 and figure 29 indicate the possible API calls that would be suitable for the Throttle Controller and the Adaptation Controller.

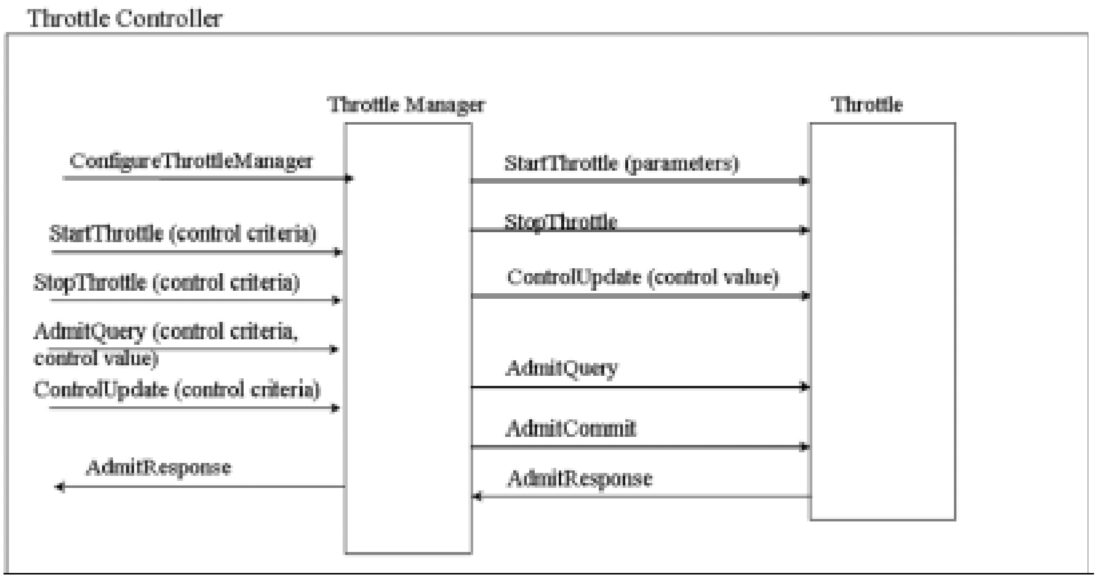
Throttle Controller
Throttle Manager
Throttle
ConfigureThrottleManager
StartThrottle (control criteria)
StopThrottle (control criteria)
AdmitQuery (control criteria, control value)
ControlUpdate (control criteria)
AdmitResponse
StartThrottle (parameters)
StopThrottle
ControlUpdate (control value)
AdmitQuery
AdmitCommit
AdmitResponse

Figure 28: Outline API for the Throttle Manager and associated Throttles

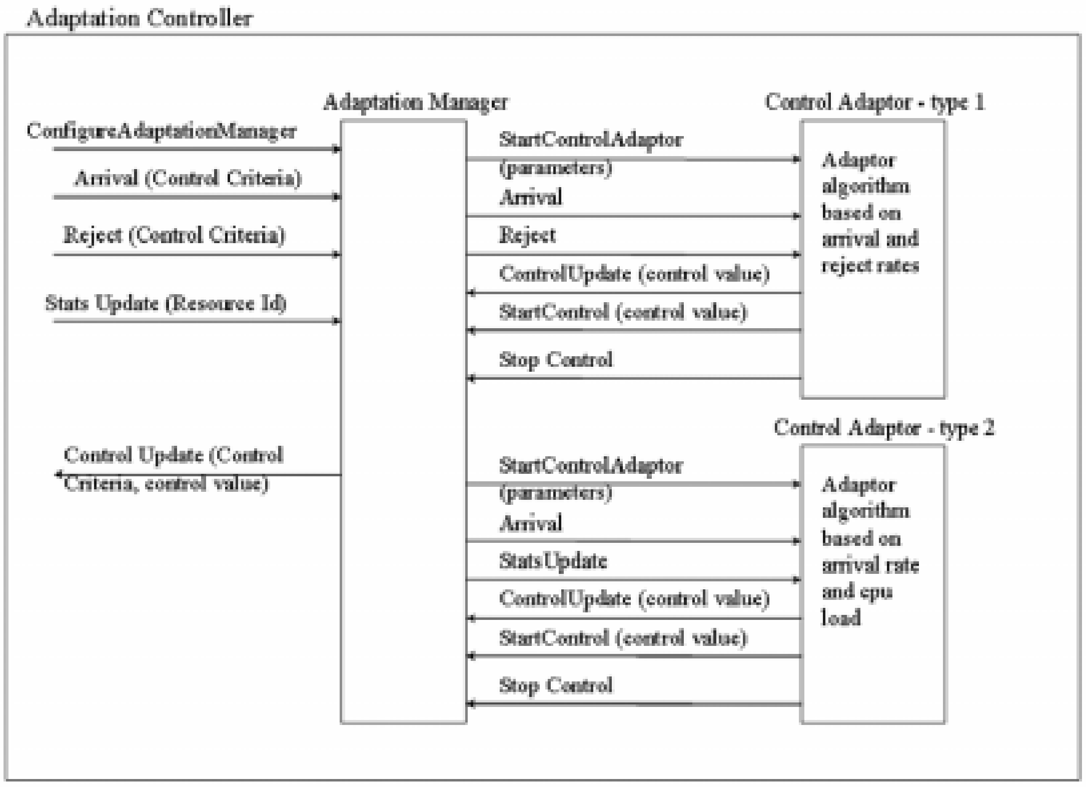
Adaptation Controller
Adaptation Manager
Control Adaptor - type 1
ConfigureAdaptationManager
Arrival (Control Criteria)
Reject (Control Criteria)
Stats Update (Resource Id)
Control Update (Control Criteria, control value)
StartControlAdaptor (parameters)
Arrival
Reject
ControlUpdate (control value)
StartControl (control value)
Stop Control
Adaptor algorithm based on arrival and reject rates
Control Adaptor - type 2
StartControlAdaptor (parameters)
Arrival
StatsUpdate
ControlUpdate (control value)
StartControl (control value)
Stop Control
Adaptor algorithm based on arrival rate and cpu load

Figure 29: Outline API for the Adaptation Manager and associated Control Adaptors

6.13 Specification of network management interfaces

The management function interacts with NOCA components in four different ways:

1) The NOCA components are instantiated and initialized on enabled equipment by operator intervention via the management system. It may be that the definition of hosts capable of supporting remote throttles, or from whom throttle requests would be accepted will be defined by the management system.

2) The management system can be used to instantiate and arm manual controls. These controls are not modified by the NOCA components, but use the NOCA infrastructure.

3) The management system collects statistics from throttles detailing the numbers of sessions accepted/rejected by NOCA throttles and the activity of Control Adapters.

4) The management system will collect alarms generated when NOCA components activate new throttles etc or when internal errors or problem arise.

6.14 Security design

Most of the functions in the NOCA are between elements within the NGN. It is not envisaged that the NOCA would extend out to the end user and so the security implications are relatively light. The key security risk associated with the NOCA is GOCAP, in which remote servers send instructions to network elements to reject service requests to particular destinations. The potential for abusing this functionality to implement denial of service attacks is obvious, especially as these control associations may cross operator boundaries. The security requirements for GOCAP are described in clause 7.3.

6.15 Mapping control components to processing cards

Server architectures may differ in the way that they map the various communication applications and the protocol stacks onto processing cards. In particular, the communication applications may only be located on some of the cards, not all. For example, processing of the lower part of the protocol stack, that is transport protocols such as UDP/IP or SCTP/IP, may be done on separate cards from the processing of the communication applications and upper layers of the stack.

The NOCA has been designed to use the following information that is available to communication applications:

- knowledge of which communication protocol (e.g. SIP, LDAP) is being used to convey a service request from source to target; Throttles are applied against messages in specific protocols;
- knowledge of which messages may be rejected (e.g. SIP INVITE, SIP REGISTER) and which must not (e.g. SIP RESPONSE, ACK, CANCEL, BYE);
- where relevant, knowledge of called address (e.g. SIP Request-URI), in order to throttle requests relating to a specific called address where appropriate;
- where relevant, knowledge of called address translation, in order to chain feedback controls in series;
- knowledge of source node identity (throttles apply at specific source nodes against specific protocols).

Consequently, the NOCA components may be located in the following ways:

- either on the processing cards that run communication applications (see figure 30);
- or on a separate NOCA server, which communicates with the processing cards that run communication applications (see figure 31).

NOTE 1: The mechanism by which NOCA cards communicate with other cards within a single server does not require standardization, it being an implementation issue for the server vendor.

Figure 30 shows a server with two cards running Communication Applications (indicated by squares) and cards not running Communication Applications (indicated by circles). It shows NOCA configured on the Communication Application cards facing outwards to external servers.

NOTE 2: GOCAP could be used internally, for example between the two Communication Application cards, as indicated.

NOTE 3: It is possible for the cards not running Communication Application to become overloaded (for example, by high, or unbalanced, signalling load). It is up to the server's vendor to decide how to cope with this situation. Possible solutions are the following:

- Provide enough processing power on the non Communication Application cards in order to force the Communication Application cards always to be the bottleneck.

- Provide a mechanism for an overloaded non Communication Application card to indicate to a Communication Application card that it is overloaded.

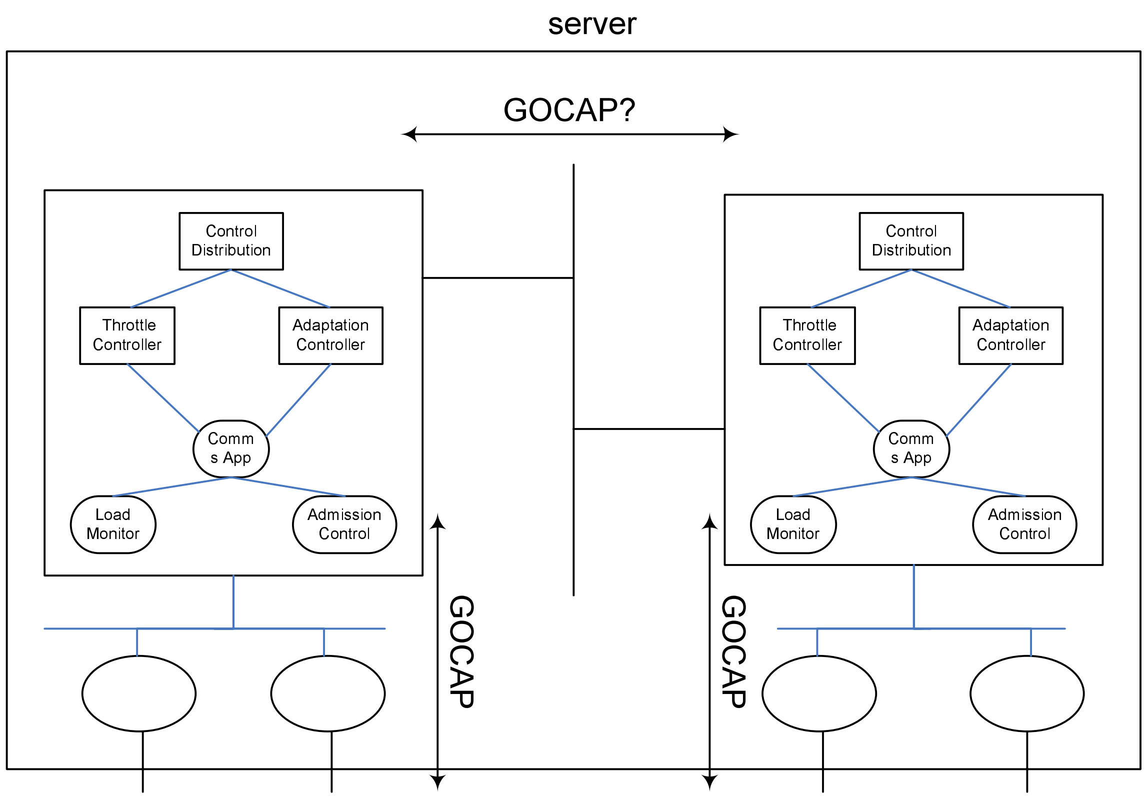

Figure 30: NOCA embedded in Communication App cards of a server

Figure 31 shows the same server as figure 30, but with GOCAP running on a separate internal GOCAP server (indicated by the square box with rounded corners). For this configuration, each Communication Application card has a GOCAP proxy to manage GOCAP related communications, and the GOCAP server card has the full set of GOCAP components.

If a Communication Application card were causing overload at an external server then the latter would send NOCA control updates to the Throttle Controller which would instantiate a throttle. The Communication Application, would then ask its proxy whether it could send a service request to the external server, and its proxy would forward the request to the GOCAP server. The GOCAP server would forward the request to the Throttle Controller, and the response would be conveyed back to the Communication Application.

If a Communication Application card were overloaded by external servers, then the card's Load Monitoring and/or Admission functions would notify the card's GOCAP proxy of high loads and/or reject events. That proxy would forward the notifications to the proxy at the GOCAP server, which would pass them to the Adaptation Controller. The latter then begins load/reject monitoring and adaptation of the control variable. Control variable updates are sent to the external server by the NOCA Control Distribution component.

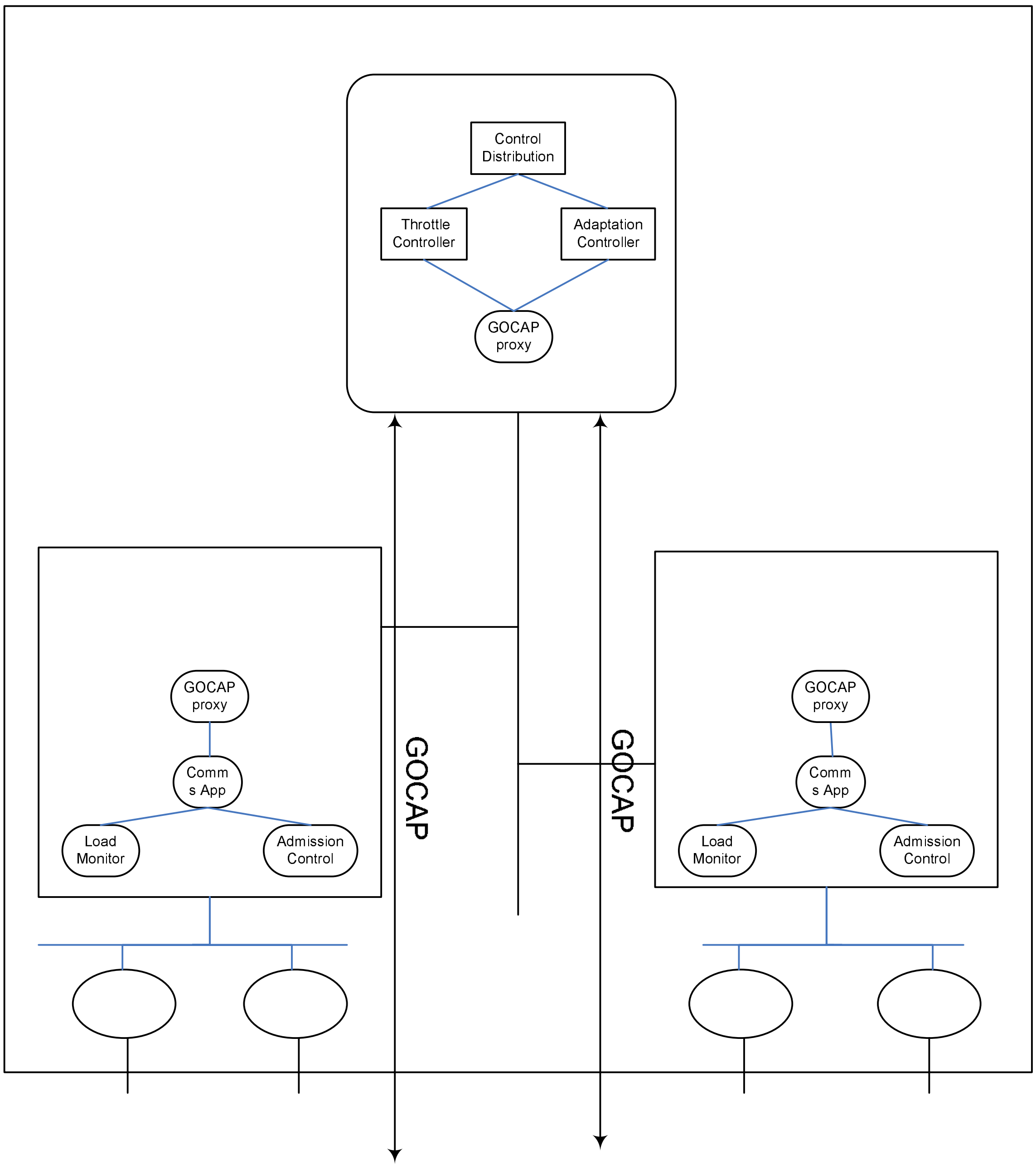

Figure 31: Server with internal NOCA card

6.16 Performance proving

Tests to prove that overload controls perform adequately fall into two main types:

Deterministic, i.e. calls are generated with constant inter-arrival times. This type of test is designed in such a way that the behaviour of the system is very precisely determined. It enables testing of whether:

- the overload control logic is working correctly;
- parameter values are being correctly assigned;
- measurements (for statistics) are being correctly calculated.

Realistic load and configuration: These types of tests are designed to determine the behaviour of the overload control when the test set-up is configured to mimic a more realistic network configuration, including the random nature of real traffic. Optimal or near-optimal parameter values would be used. It enables a determination of whether the observed behaviour sufficiently closely matches that determined by modelling (discrete event simulation), and hence whether the code implements the end-to-end overload control design faithfully.

In each of the above two cases, it is the end-to-end behaviour of the control that is being tested, and therefore such tests should cover both the overload detection and the restriction schemes.

The following testing and modelling facilities will be required:

- call generators able to mimic the call state machine at source nodes, including the relevant parts of the overload control (if real servers are not available);
- equipment able to measure response times and calling rates (before and after restriction at source nodes, and admitted and rejected at the target node) on a second-by-second basis - required to understand the fast dynamics of the end-to-end control of powerful servers under overload;
- very high calling-rate generation (for example several thousand calls per second) to overload large capacity servers;
- a realistic model of the end-to-end overload control (usually a detailed simulation) - required to check that all parameter values cope well over the required range of overload scenarios;
- a comparison of test measurements with the simulation model's results - required if the full range of overload scenarios cannot be generated in a test facility.

7 GOCAP requirements

7.1 Introduction

Whereas clause 6 describes the nodal behaviours required to implement a distributed processing overload scheme, clause 7 describes the requirements for the Generic Overload Control Application Protocol (GOCAP). The GOCAP is proposed to define an optional parallel interface to an existing control interface which supports the real time management of dynamic transaction request peaks between components in or interfacing to the control plane of a NGN. The use of such functionality removes the unpredictable and undesirable degradation of such components, allows systems to maintain their real-time processing profile whilst maximizing the effective transactions.

The relationship of a generic overload control interface between architectural entities is represented below (see figure 32) where an existing control interface can optionally have a parallel generic overload control interface. This approach in no way changes the control interface *C*a, rather the overload control interface is used to affect the behaviour of the client. For simplicity the model shows a client / server relationship, with the request source as the client and the target as the server, but it should be noted that on many control interfaces the demand can be sourced by both architectural entities, i.e. in both directions so that either end can be simultaneously acting as both a client and a server.

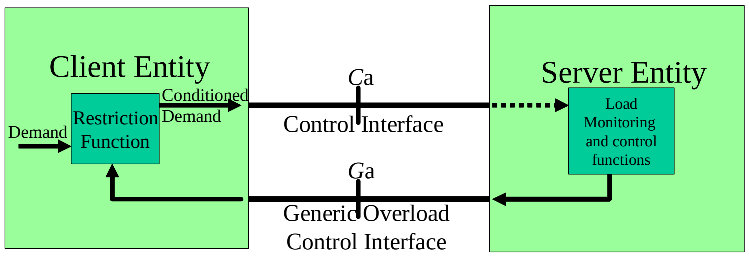

Figure 32: Relationship of generic overload control interface

7.2 Transport requirements

GOCAP is aimed at supporting real time applications and therefore requires the following network transport characteristics:

a) GOCAP requires a reliable transport mechanism so that crucial overload control messages are not lost at the very time when the control requests are causing the server to approach overload. (High demand peaks can occur during conditions such as control packet or even media stream packet loss, as resultant application misbehaviour often results in multiple user repeated attempts.)

b) GOCAP requires a transport mechanism that does not introduce significant delay that would affect the speed at which a control was applied to a source.

c) A transport layer failure may precipitate server overload, so it is important that GOCAP sessions should be able to survive such failures. The GOCAP transport mechanism should, therefore, be resilient or capable of fast restoration.

It is recommended the use of SCTP multi-homing associations should be at least an option for meeting these requirements.

7.3 Security requirements

GOCAP, by its very nature, can intrinsically be used for Denial of Service (DoS) attacks from just a single source spoofing a control client. Native GOCAP is susceptible to Masquerade and "Man-in-the-Middle" attacks, (as are the same mechanisms built into existing protocols). Therefore there shall be an option for underlying transport mechanisms to provide authentication of the transport end-points and privacy of the information flows. The special security issues related to the use of GOCAP between different network operators also needs to be taken into account. In particular it shall not be possible for one operator to cause restriction of request flows that do not terminate or transit that operators network.

It is recommended that the defined GOCAP transport profiles include mandatory options for transport links such as TLS or IPsec.

7.4 General requirements

7.4.1 Overload of GOCAP dialogues

In GOCAP deployments it shall be assumed that:

a) the priority given by client and server entities to processing GOCAP messages is at the expense of processing the control protocol messages;

b) consequently the proportion of the server or client entity processing used is a significantly small percentage of that available and therefore no overload controls are required at the GOCAP level.

7.4.2 Administrative requirements

7.4.2.1 General

a) Multiple GOCAP dialogues from different clients to one server shall be supported.

b) Multiple control protocols from a single client to a server shall be supported via multiple GOCAP dialogues.

c) GOCAP shall support multiple overload control methods and be extensible for new methods to be added in latter versions.

7.4.2.2 Version identification

a) All GOCAP messages shall identify the current protocol version.

b) GOCAP dialogues shall convey information on the control protocol and its characteristics to ensure that it the parallel control stream is unambiguously associated with the GOCAP dialogue.

7.4.3 GOCAP dialogues

7.4.3.1 GOCAP dialogue establishment

a) For a control protocol that registers one entity with the other entity, the registrar shall default to the GOCAP Master and the source of registration request the GOCAP Slave.

b) For control protocols that do not register one entity with the other, each entity's management system shall be capable of designating which entity is the Master and which is the Slave.

c) All Master / Slave designations shall only have scope for the individual dialogue between two entities. E.g. for three entities in a linear control chain, the middle entity must be able to be a Master to one entity and simultaneously be a Slave to the other if required.

d) GOCAP Masters shall be responsible for invoking the opening of the appropriate transport link to the Slave. If an attempt to establish a transport link fails, it will be reattempted after a configurable time interval.

e) The Master entity's management system shall attempt to establish a transport link a configurable number of times, before reporting a failure back to the entity's management system. When this retry count is reached it shall be reset and link establishment attempts shall continue.

f) Once the transport path has been established between the two entities a GOCAP dialogue shall be established by the designated Master.

g) Each entity shall report any GOCAP dialogue establishment failure to its management system.

h) To facilitate interoperability and upgrades, the GOCAP dialogue establishment shall negotiate the GOCAP version to be used.

i) GOCAP dialogue establishment shall agree if the overload control relation is from Master to Slave only, Slave to Master only or if bi-directional control is allowed for the supported protocol.

j) GOCAP dialogue establishment shall exchange the Control Protocol name and version to which it refers. A mismatch of Control Protocol name and version shall result in a dialogue failure.

k) GOCAP dialogue establishment shall exchange the overload control method being used (e.g. gapping, percentage call restriction, scaled restriction rates, etc). A mismatch of overload control mechanism shall result in a dialogue failure.

l) GOCAP dialogue establishment shall exchange the type address / request reference to which restriction requests will be applied, e.g. ITU-Recommendation E.164 [10], national/private telephone numbers, URIs, etc.

m) GOCAP dialogue establishment shall exchange the Control Protocol "end point identifiers" to which it refers. If these identifiers do not match those expected for the parallel control path, this shall result in a dialogue failure.

n) A GOCAP Master shall poll its Slave with a request to open a dialogue. The time between of poll attempts shall be configurable by the management system.

o) The GOCAP Master entity's management system shall support configuring the number of failed poll attempts allowed, before reporting a failure back to its management system.. In event of failure, this counter is reset and polling continued. Subsequent failures are to be reported to the management system. Once a dialogue is established, the server shall continue and report success to the management system.

p) Each request to open a dialogue shall be identifiable by a sequence number and all responses shall use this to identify the associated request.

q) If a dialogue exchange has occurred but fails to negotiate a dialogue (e.g. the Master and Slave cannot agree the GOCAP version to use), the Master shall halt polling until restarted by an action from its management system. Such a situation would require manual intervention and reconfiguration and therefore the GOCAP establishment shall cease pending a management restart.

r) If the underlying transport layer reports that the data connection is lost, the GOCAP Master shall automatically attempt to re-establish the transport path and then the GOCAP dialogue as described above.

s) A Master shall open up a separate GOCAP dialogue for each control protocol supported even if a dialogue has already been established to the same client for a different control protocol.

7.4.4 GOCAP dialogue status monitoring

a) When a dialogue has been successfully established, both entities in a dialogue shall poll the other with a status request to ensure that the overload control process is active at the other end. The poll rate shall be individually configurable per GOCAP dialogue by each entities own management system. The time from dialogue establishment and the first poll shall be randomly chosen within the configured polling period.

b) Each status request shall be identifiable by a sequence number provided by the source of the request and all responses shall use this to identity the associated request.

c) When a status request is acknowledged the normal poll cycle shall continue but if a status response is not received within some configurable interval, the source entity shall repeat the status request. If a response is not received after a certain number of repeat requests, the GOCAP dialogue shall be deemed to be lost and a new dialogue initiated. Loss of GOCAP dialogue shall be reported to the management entity.

d) Loss of the underlying transport link shall be indicated to the GOCAP Master which shall automatically attempt to re-establish the transport connection.

7.4.5 GOCAP restrict message requirements

a) When an overload is detected by the load monitoring function within the control protocol server, a restrict request shall be issued.

b) All restrict requests shall be identified by a sequence number so as to identify repeat requests.

c) All restrict requests shall have a duration validity field which specifies the default number of seconds the restriction is valid unless updated with a new restriction message. Subsequent restriction requests with identical restriction criteria shall overwrite the duration field to that in the new request.

d) If the client entity in unable to accommodate a restriction request, it shall indicate this to the server via a negative acknowledgement. Negative acknowledgements shall be notified to both GOCAP Slave and Master entities management systems.

NOTE 1: Positive acknowledgements may not be required if the transport can indicate to GOCAP delivery failure.

e) Negative acknowledgement shall indicate what aspect of the request could not be accommodated.

NOTE 2: Specific negative acknowledgements may minimize the work done in constructing and sending positive acknowledgement.

f) If a restrict request fails to be delivered, resending will be attempted a number of times after which the failure shall be notified to the requesting entity's management system.

g) The control protocol server shall send restriction requests that shall be applied to:

1) all destination addresses;

2) partial destination addresses;

3) an individual destination address.

h) All TISPAN NGN address formats shall be supported.

i) As appropriate to the type of address, individual characters shall be capable of being wild carded.

EXAMPLE 1: For a wildcard character "?" an E.164 number might be represented by 199236367?1. This would apply only to 19923637611, 19923636721, 19923636731, …………, 19923636791.

j) As appropriate to the type of address, individual address fields should be capable of being wild-carded.

EXAMPLE 2: For a wildcard character "*", a URI with IP address might be restricted with *telepromotion.bigco@10.153.42.**. This would apply to all session requests to the telepromotion.bigco at the sub-net 10.153.42.

k) For E.164 and telephone numbers it shall be possible to wildcard digit at the end of a number string.

EXAMPLE 3: For a wildcard character of "*", then 44147360* would apply to addresses strings where the first eight digits matched the restriction string, i.e. 441473603377 would be a match.

l) It shall be possible to up apply simultaneous restrictions where address wild-carding overlaps if one address can be determined as more specific than another. In this situation the more specific address type will have priority.

EXAMPLE 4: For telephone numbers this means that longest length has priority so that a restriction to 14023391182? would take precedence over 140233911*.

7.5 GOCAP example Meta-protocol

7.5.1 Dialogue establishment messages

7.5.1.1 Open dialogue message

The following psuedo-protocol is an expression of the requirements for an OPEN dialogue message, initiated by the GOCAP Server.

```
GOCAP:      Version n.m     ;Client GOCAP version
Message:    OPEN
Server Ref: string          ; reference unique to the Server
Client Ref:                 ; null
Svr Seq No: nnnnnnnn        ;
Clnt Seq No:                ; null
Versions:   n.m n.m ….. n.m ; list of version numbers supported
Cntrl Type: (S->C | C->S | S<-> C)  ; Server to Client or Client to Server or bi-directional control
                                    ; i.e. overload control commands issues from server to
                                    ; client only or from both Server and Client
Cntrl Prot: string n.m      ; name as defined in the Protocol Classification register & version
OvlCtl Type:  Gap | Perc | etc  ;type of overload control mechanism supported
Server Id:  string          ; identifier of Server as used in Control Protocol
Client Id:  string          ; identifier of Client as used in Control Protocol
```

7.5.1.2 Open dialogue acknowledgement message

The following pseudo-protocol is an expression of the requirements for an OPEN ACKNOWLEDGE dialogue message in response to an OPEN request. The fields are checked by the client have an OK or FAIL indication added to them. A dialogue shall only be deemed to be successful if all checked fields indicate "OK".

```
GOCAP:      Version n.m
Message:    OPEN_ACK                ; dialogue open acknowledgement
Server Ref: string                  ; Server Ref as in associated OPEN request
Client Ref:                         ; null
Svr Seq No: nnnnnnnn                ; Sequence No. as in associated OPEN request
Clnt Seq No:                        ; null
Versions:   n.m (OK | FAIL)         ; OK =version number to be used, FAIL=no version match
Cntrl Type: (S->C | C->S | S<-> C) (OK | FAIL)
                                    ; Control Type as in the associated OPEN request.
Cntrl Prot: string n.m (OK | FAIL)  ; name and version as defined in the
                                    ; Protocol Classification register + version number
```

```
OvlCtl Type:Gap (OK | Fail) | Perc (OK | Fail) | etc
                                ;type of overload control mechanism supported
Server Id:  string  (OK | FAIL)     ; identifier of Server as used in Control Protocol
Client Id:  string  (OK | FAIL)     ; identifier of Client as used in Control Protocol
```

7.5.2 Dialogue status messages

The following psuedo-protocol is an expression of the requirements for dialogue status request and response messages.

7.5.2.1 Status message originated by a server

```
GOCAP:      Version n.m     ; negotiated GOCAP version
Message:    STATUS          ;
Server Ref: string          ; reference unique to the Server
Client Ref:                 ; null
Svr Seq No: nnnnnnnn        ;
Clnt Seq No:                ; null
```

7.5.2.2 Status message originated by a client

```
GOCAP:      Version n.m     ; negotiated GOCAP version
Message:    STATUS          ;
Server Ref:                 ; null
Client Ref: string          ; reference unique to the Client
Svr Seq No:                 ; null
Clnt Seq No:nnnnnnnn        ;
```

7.5.2.3 Status acknowledge message in response to a request originated by a server

```
GOCAP:      Version n.m     ; negotiated GOCAP version
Message:    STATUS_ACK      ;
Server Ref: string          ; reference unique to the Server
Client Ref:                 ; null
Svr Seq No: nnnnnnnn        ;
Clnt Seq No:                ; null
```

7.5.2.4 Status acknowledge message in response to a request originated by a client

```
GOCAP:      Version n.m     ; negotiated GOCAP version
Message:    STATUS_ACK      ;
Server Ref:                 ; null
Client Ref: string          ; reference unique to the Client
Svr Seq No:                 ; null
Clnt Seq No:nnnnnnnn        ;
```

7.5.3 Overload restriction messages

The following pseudo-protocol is an expression of the requirements for overload restriction request and response messages.

7.5.3.1 Restrict request message

```
GOCAP:      Version n.m     ; negotiated GOCAP version
Message:    RESTRICT        ;
Server Ref: string | null   ; null if initiated from client
Client Ref: string | null   ; null is initiated from server
Svr Seq No: string | null   ; null if initiated from client
Clnt Seq No:string | null   ; null if initiated from server
Class:      nn              ; Session type class number as defined in the
                            ; Protocol Classification register
Param:  p.qr |40%|etc       ; restriction parameter depending on the negotiated restriction
                            ; type e.g. for gapping it would be the gap time
                            ;in seconds (to 2 decimal places), for proportional restriction it
                            ; would be a percentage of sessions admitted, etc
                            ; shortest window to allow a new session origination
Adrs Type:  string          ; type of address in the Adrs field e.g. E.164, tel No., URI,
Adrs:       string          ; address range to which restriction applies
Duration:   60->3600        ; No. seconds the restriction applies unless updated.
```

7.5.3.2 Restrict negative acknowledgement message

```
GOCAP:       Version n.m             ; negotiated GOCAP version
Message:     RESTRICT_NACK           ;
Server Ref: string | null            ; null if initiated from client
Client Ref: string | null            ; null is initiated from server
Svr Seq No: string | null            ; null if initiated from client
Clnt Seq No:string | null            ; null if initiated from server
Param:       p.qr |40%|etc (OK|FAIL) ; OK = Accepted, Fail = number of restriction conditions
                                     ;full in source limiter
Adrs Type:   string (OK | FAIL)      ; address type OK= recognized, FAIL = unrecognised
Adrs:        string (OK | FAIL)      ; address string or wild-carding
                                     ; OK= recognized, FAIL = unrecognised
```

Annex A: Example information flows

To make clear some of the logic described above, the following message flows demonstrate the information being passed between the different NOCA entities when NOCA is being used to protect against server overload. Figure 33 shows a simple session in which application App1 on Server1 sends a request to App2 on Server2. To service this request requires App2 to consult application running on Server3 and Server4. Server2 has local admission control (as part of App1). The message flow shows how this simple request is handled.

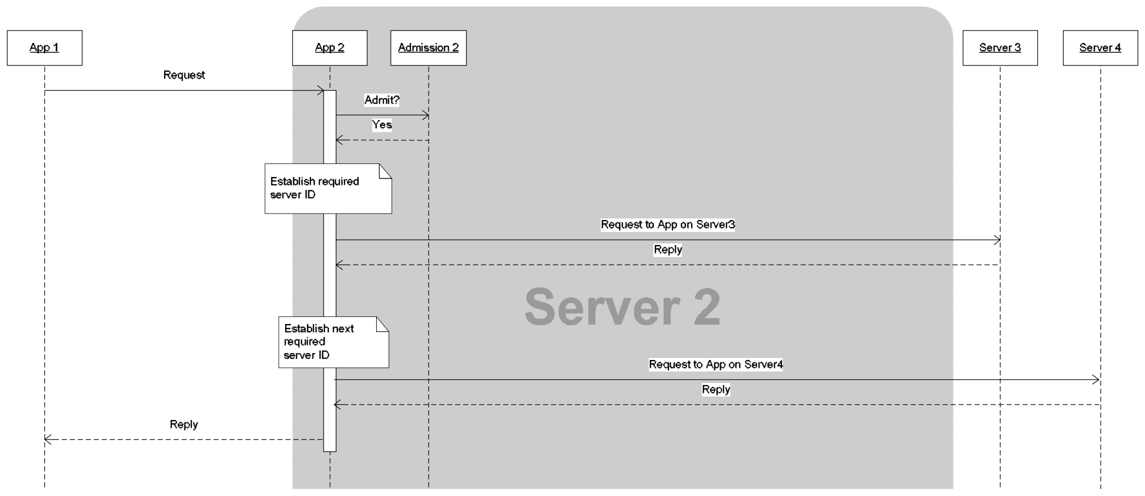
App 1
App 2
Admission 2
Server 3
Server 4
Request
Admit?
Yes
Establish required server ID
Request to App on Server3
Reply
Server 2
Establish next required server ID
Request to App on Server4
Reply
Reply

Figure 33: A simple session

Figure 34 shows the simple request in an environment in which NOCA is deployed. In this case, there are no NOCA throttles instantiated (none of the servers in overloaded). Of course App1 and App2 do not know that there are no active throttles, which is why each request is offered to the throttle controller even though there are no active throttles.

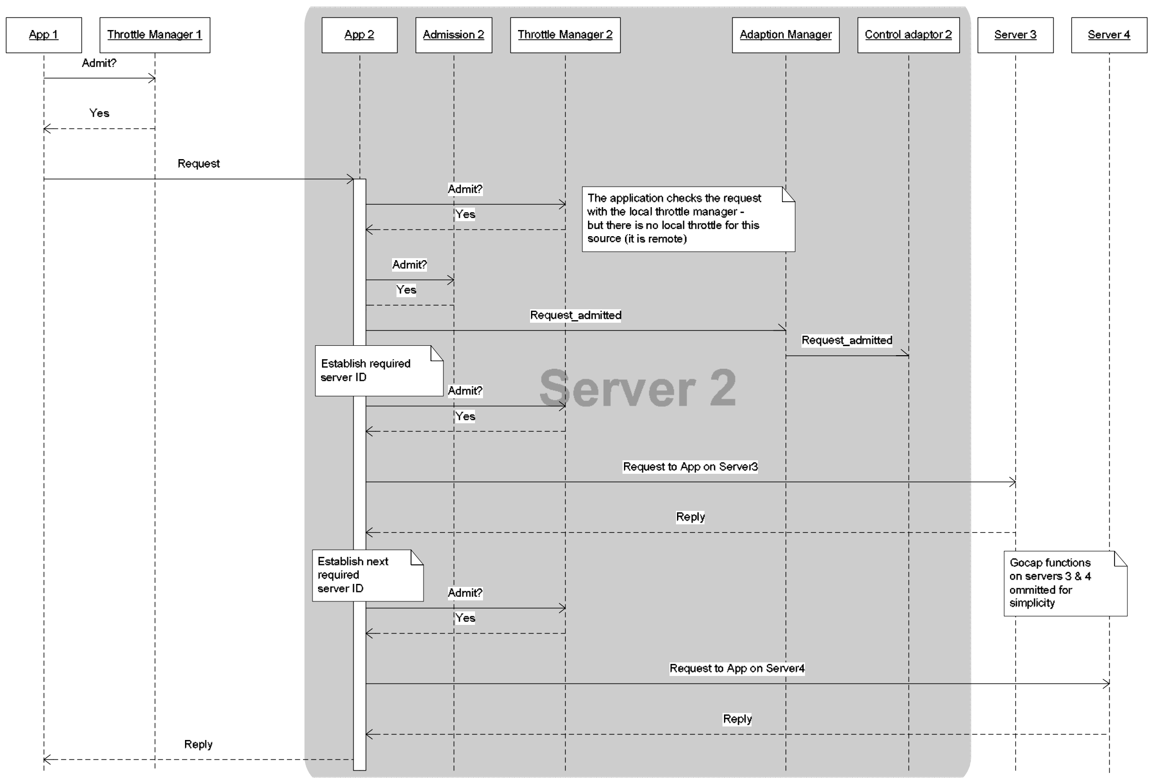
App 1
Throttle Manager 1
App 2
Admission 2
Throttle Manager 2
Adaption Manager
Control adaptor 2
Server 3
Server 4
Admit?
Yes
Request
Admit?
Yes
The application checks the request with the local throttle manager - but there is no local throttle for this source (it is remote)
Admit?
Yes
Request_admitted
Request_admitted
Establish required server ID
Server 2
Admit?
Yes
Request to App on Server3
Reply
Establish next required server ID
Admit?
Yes
Gocap functions on servers 3 & 4 ommitted for simplicity
Request to App on Server4
Reply
Reply

Figure 34: A simple call in a GOCAP enabled system with no overload

When the systems are overloaded, NOCA throttles will be instantiated. Figure 35 shows a successful session when Server2 and Server3 are both overloaded.

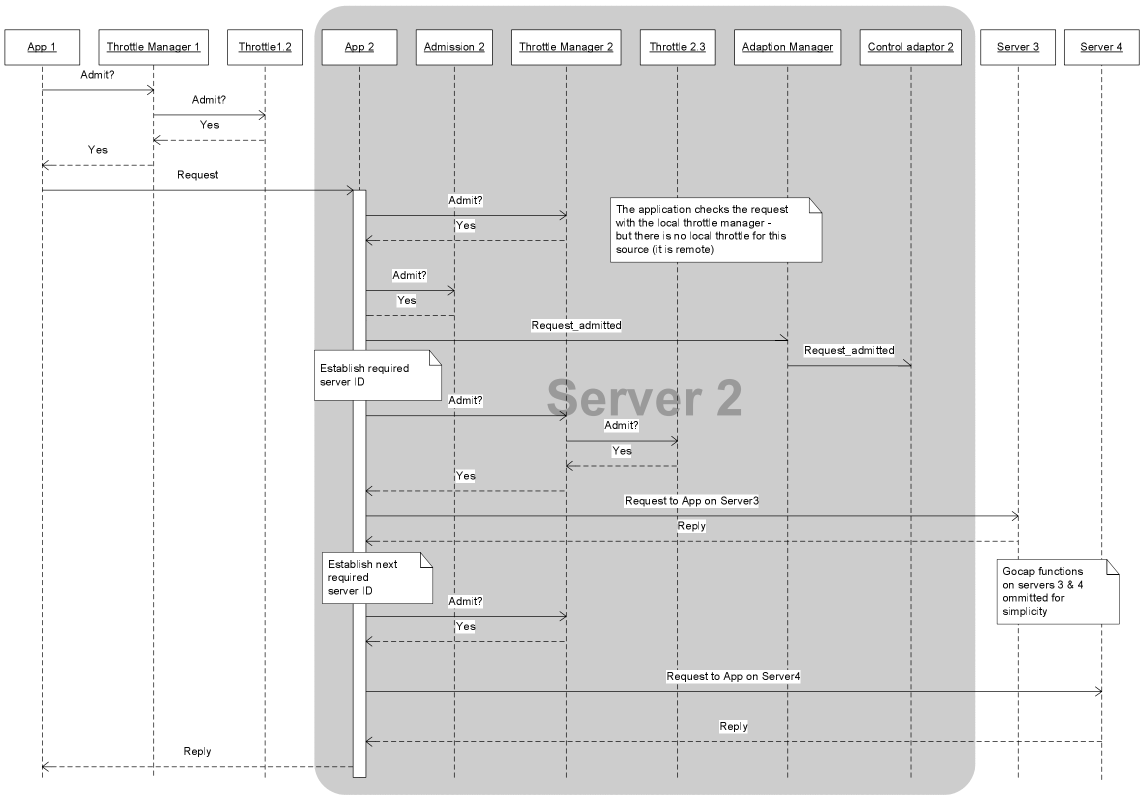
App 1
Throttle Manager 1
Throttle1.2
App 2
Admission 2
Throttle Manager 2
Throttle 2.3
Adaption Manager
Control adaptor 2
Server 3
Server 4
Admit?
Admit?
Yes
Yes
Request
Admit?
Yes
The application checks the request with the local throttle manager - but there is no local throttle for this source (it is remote)
Admit?
Yes
Request_admitted
Request_admitted
Establish required server ID
Server 2
Admit?
Admit?
Yes
Yes
Request to App on Server3
Reply
Establish next required server ID
Gocap functions on servers 3 & 4 ommitted for simplicity
Admit?
Yes
Request to App on Server4
Reply
Reply

Figure 35: A successful session during overload on Server2 and Server3

We can see in figure 36 how Server2 is protected by the remote throttle on the source.

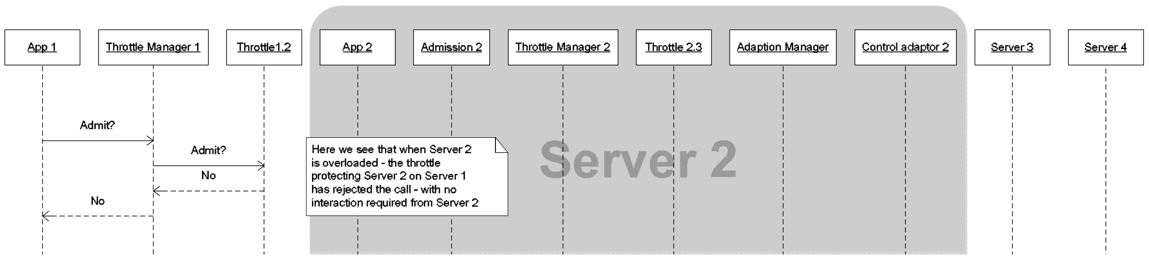
App 1
Throttle Manager 1
Throttle1.2
App 2
Admission 2
Throttle Manager 2
Throttle 2.3
Adaption Manager
Control adaptor 2
Server 3
Server 4
Admit?
Admit?
No
No
Here we see that when Server 2 is overloaded - the throttle protecting Server 2 on Server 1 has rejected the call - with no interaction required from Server 2
Server 2

Figure 36: A session rejected by the NOCA throttle at Server1 protecting Server2

Figure 37 shows how the overloaded server, Server3, is protected by the throttle on Server2. Notice that the NOCA throttle for Server3 should be consulted as soon as Server3 is identified as supporting the session as that helps to reduce the processing load on Server2. In this case it is assumed that the session is rejected if throttled, but App2 could be configured to pass the request onto another server.

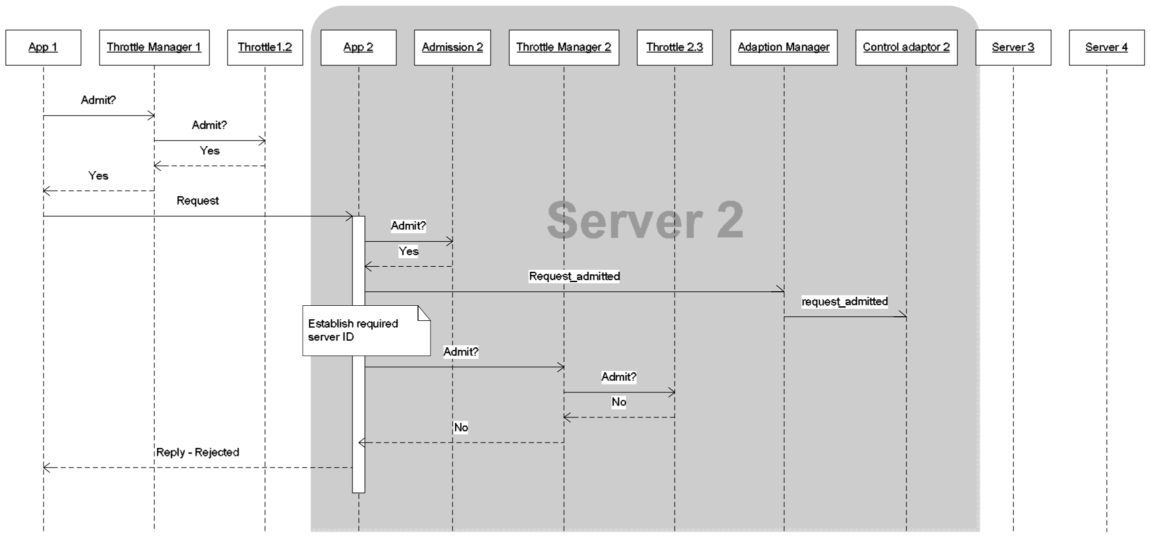

Figure 37: A session rejected by the GOCAP throttle protecting Server3

In figure 38 shows a GOCAP interworking scenario, where the source is not GOCAP aware. This means that that there is no NOCA throttle on the source, and requests arriving at Server2 are not controlled. Server2 is overloaded and has deployed an admission throttle (Throttle2.2) to regulate the arrivals from the source. Obviously, a local throttle does not protect the host any more than local admission control, but NOCA allows them as it enables the controlled apportionment of processing resource between competing request sources and reduces the unfair throttling of those sources that do support overload control.

Figure 39 shows a request rejected by the local NOCA throttle. These locally rejected requests are treated as if rejected by App1 remotely as far as the NOCA Adaptation Manager is concerned, and so the local admission control decision is not passed to Adaptation Manager and does not contribute to the target rejection rate of the system.

Finally, figure 40 shows a request rejected by the local admission control, and this rejection information is passed to the Adaptation Manager.

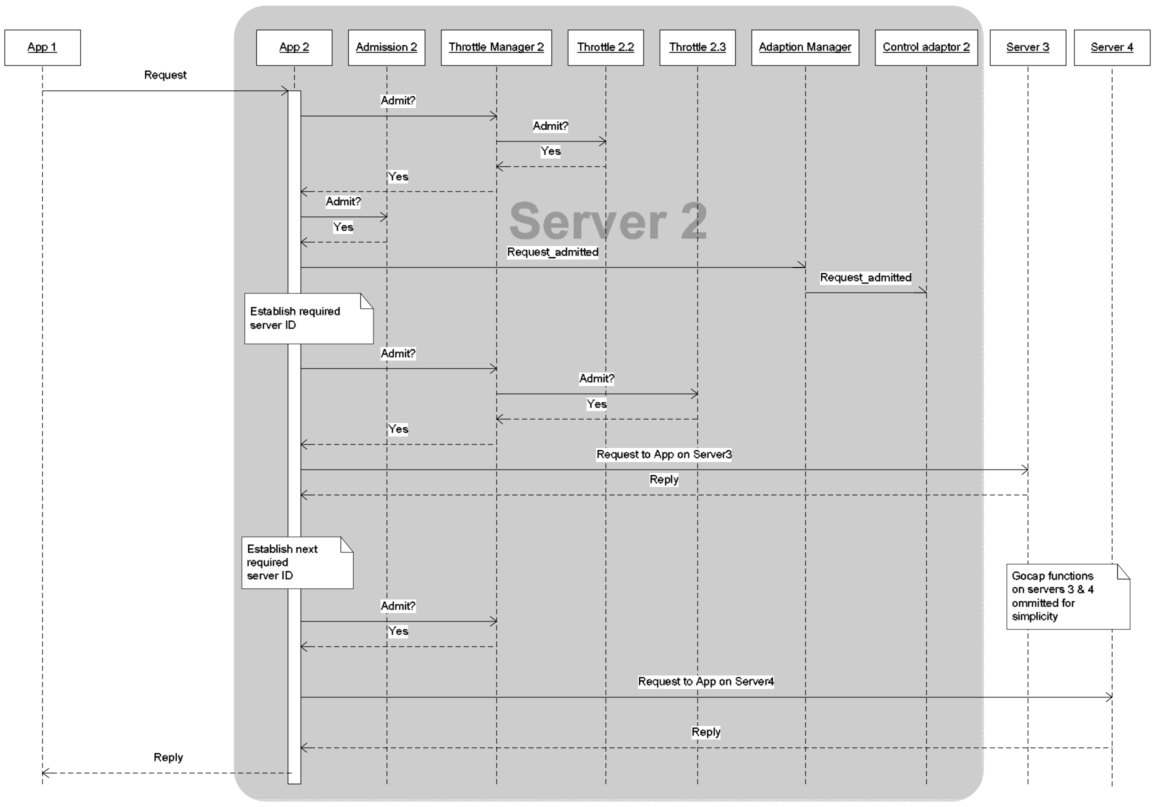

Figure 38: A successful session using a local throttle on Server2

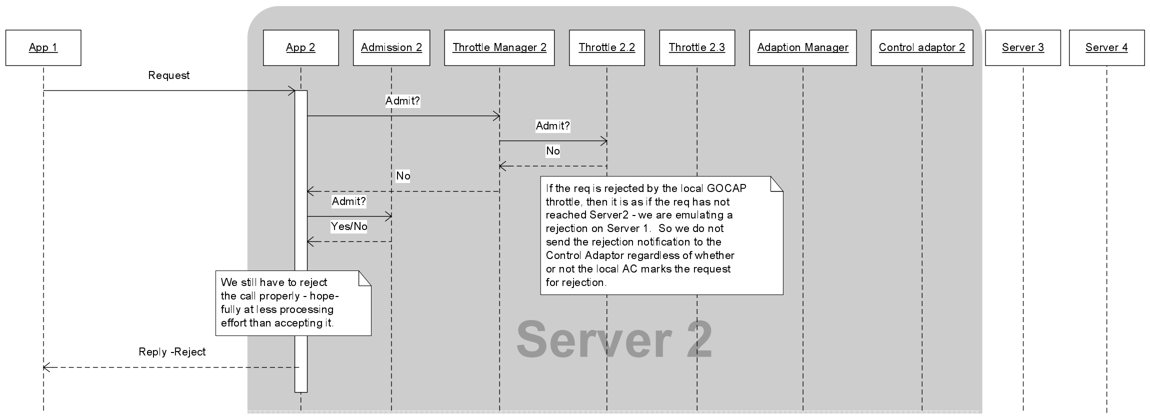
App 1
App 2
Admission 2
Throttle Manager 2
Throttle 2.2
Throttle 2.3
Adaption Manager
Control adaptor 2
Server 3
Server 4
Request
Admit?
Admit?
No
No
Admit?
Yes/No
If the req is rejected by the local GOCAP throttle, then it is as if the req has not reached Server2 - we are emulating a rejection on Server 1. So we do not send the rejection notification to the Control Adaptor regardless of whether or not the local AC marks the request for rejection.
We still have to reject the call properly - hope-fully at less processing effort than accepting it.
Server 2
Reply -Reject

Figure 39: Rejection of a session request from Server1 by local throttle at Server2

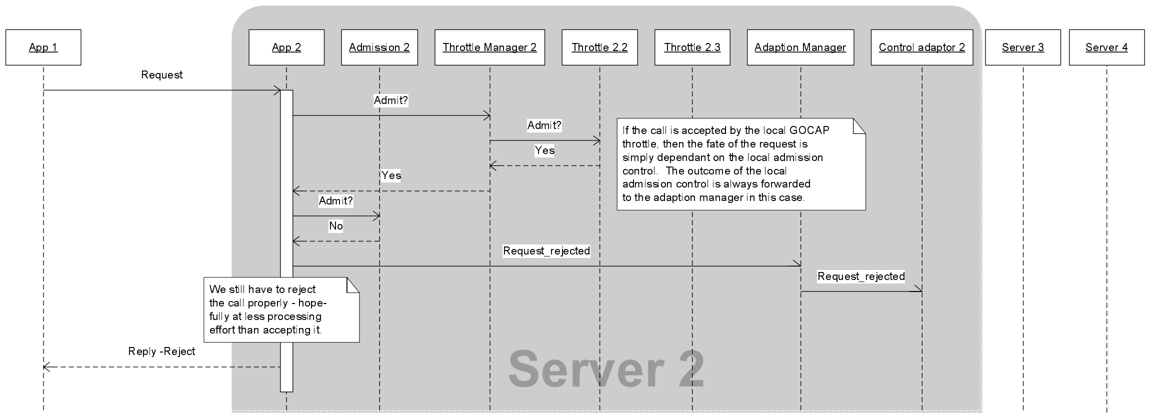
App 1
App 2
Admission 2
Throttle Manager 2
Throttle 2.2
Throttle 2.3
Adaption Manager
Control adaptor 2
Server 3
Server 4
Request
Admit?
Admit?
Yes
Yes
Admit?
No
Request_rejected
Request_rejected
If the call is accepted by the local GOCAP throttle, then the fate of the request is simply dependant on the local admission control. The outcome of the local admission control is always forwarded to the adaption manager in this case.
We still have to reject the call properly - hope-fully at less processing effort than accepting it.
Reply -Reject
Server 2

Figure 40: Rejection of a session request by App2 admission control at Server

History

Document history		
V1.1.1	October 2006	Publication