Methods to synchronize the IN SCP's overload protection mechanism

Philip Ginzboorg

Nokia Research Center

P.L. 45, 00211 Helsinki, Finland

E-mail: philip.ginzboorg@research.nokia.com

Abstract

The Intelligent Network's (IN) architecture is based on Service Switching Points (SSP's) that switch telephone calls, and the Service Control Points (SCP's) that make part of the decisions about the routing and the charging of calls that are switched by the SSP's. To communicate, SSP and SCP use the IN Application Part (INAP) protocol. IN has a distributed mechanism for protecting the SCP from overload that uses a traffic filtering method called "call gapping". The synchronization of the load protection in IN is a problem because there is no reliable synchronization scheme in the INAP protocol. In the text I examine several solutions to the problem. Though formulated in terms of IN, the synchronization methods apply to any similar network, i.e. a network with a star topology, traffic throttles in the peripheral nodes, and a bottleneck in the central node.

1 SERVER'S OVERLOAD IN A CLIENT-SERVER SYSTEM

Let us start with the following example. Figure 1 shows two machines – a server and a client, that are connected by a signaling link. The server contains a database, and the client asks questions from the server by sending messages over the link. When the server receives a question it starts a transaction that results in an answer to that question; when the answer is ready, the server sends it to the client. If the server hasn't answered a question within 30 seconds the client either aborts the question, or tries to ask it again. Each answered question costs the client a penny.

If the server would have been an omnipotent machine, it would have answered every question immediately, and the plot of the answers rate against the questions rate would have looked like figure 2 a). Of course there is a limit to the amount of answers that the server can supply per a unit of time. Taking this into account, the response curve of the server should look like

figure 2 b). However, that curve is still too good to be true. In reality, when the questions rate exceeds a certain threshold for an extended amount of time, the server becomes overloaded. When the server is overloaded more and more questions will result in less and less answers, as figure 2 c) shows. Such a degradation is caused by system resources being wasted. For example, the machine allocates more and more of the free memory for buffering the questions, so there is less and less working memory for computing the answers.

Figure 1 Two machines - a client and a server - are connected by a signaling link. The server has a database, and the client asks questions from the server. The communication protocol between the machines consists from the "question", "answer" message pair.

The threshold questions' rate is not fixed. Its value depends on how much of the machine's capacity can be dedicated to answering questions. For example, the threshold will be lower than usual during the updates of the server's database. The goal of any overload protection method is to change the response curve of the real machine so, that it will resemble the ideal curve in figure 2b). The closer to the ideal, the better the protection method. In the following paragraphs I describe a method in which the work of protecting the server from the overload is partly done in the client.

Suppose that to protect itself, an overloaded server sends a "filtering request" message to the client. The request contains two parameters: the upper bound on the questions rate, u [questions/s], and the duration of the filtering. When the client receives that message, it starts to filter the questions traffic so, that its rate will be at most u ; part of the questions are aborted with no answer, without ever reaching the server. The filtering lasts for the duration specified in the request message. While the client is filtering the questions traffic it can receive another filtering request. In this case the bound rate will be updated, and the filtering duration will be extended.

When using this protection scheme, the server has two difficulties. The first difficulty is how to choose the filtering parameters. Long duration and low u mean less questions and thus less money for the server. Too short duration and too high u may not reduce the traffic enough to protect the server from the overload. A natural first step in solving that difficulty is to divide the response characteristic into regions that are called “overload levels”; within each level the filtering parameters are constant (see figure 3) If at all times the server is able to determine the overload level of itself, then the operator who is responsible for the set-up of the machine can store the filtering parameters in a “level : duration, u ” format. This is not a complete solution because the really difficult part of the problem is delegated to the operator. With a complete solution, the choice of filtering parameters should be fully automatic. In [1] Kumar describes an algorithm for periodically adjusting the parameters, based on the occupancy measures of the server.

Figure 2 Response curves of the omnipotent, the ideal, and the real machines. a) The omnipotent machine will answer every question immediately, no matter how high is the questions per

second rate. b) The ideal machine has an upper bound on the answers rate. When the questions rate exceeds that bound, the answers rate stays constant; some of the questions remain unanswered. The ratio of the unanswered to the total amount of questions is called a "grade of service". c) Up to a certain threshold, the response curve of the real machine is close to the ideal. When the questions rate exceeds that threshold the machine becomes overloaded. In the overload state more questions result in less answers.

Figure 3 The response characteristic is divided into levels. Within each level the filtering parameters are constant.

Once the filtering parameters have been chosen, the second difficulty of the server is when to send, and when not to send filtering requests. The server should send the first request when it is close to becoming overloaded. After that the server should send a new request either when the duration expires, or when the filtering parameters change. The server shouldn't send a new request if the client filters questions correctly, with the right bound rate and with the right duration.

In our communication protocol, there is no feedback to the filtering request. After sending the request the server doesn't know if and how the client filters the questions traffic. If the client is the only source of questions, the server can solve the problem by monitoring the questions rate; when the monitored rate exceeds u , the server sends a new filtering request. If there are several machines that ask questions, monitoring the traffic from each needs a lot of book-keeping.

The second difficulty of the server is a synchronization problem. The server has to keep up to date, or synchronize a traffic valve that is in another, remote machine. Synchronization problems arise between a group of processes that communicate solely via message passing. A good treatment of synchronization in distributed computing is in Lamport's [2], and Jefferson's [3] papers.

The overload protection in the Intelligent Network (IN) works in a way that is very similar to the above example. In IN, the Service Control Point (SCP) is the server, and the Service

Switching Point (SSP) is the client. Synchronization of the overload protection in IN is difficult because there is no reliable synchronization scheme in the SCP-SSP communication protocol. In this text I describe several possible solutions to that problem.

Subsections 2.1 and 2.2 are a brief introduction to the IN and to the SCP's overload protection. Subsections 2.3 and 2.4 describe the difficulties in synchronizing the overload protection mechanism. The rest of the text is about how to solve the problem.

2 SCP'S OVERLOAD CONTROL IN THE INTELLIGENT NETWORK

2.1 IN architecture

The Intelligent Network's (IN) architecture is based on Service Switching Points (SSP's), and the Service Control Points (SCP's). SCP's and SSP's are connected via a Signaling System number 7 (SS7) network (figure 4) To communicate, SSP and SCP use the IN Application Part (INAP) protocol. In the SS7 protocol stack, the INAP layer is on top of the Transaction Capabilities Application Part (TCAP) layer (figure 5).

Figure 4 The Intelligent Network's architecture is based on Service Switching Points (SSP's) that switch the calls traffic, and the Service Control Points (SCP's) that make some of the decisions about the routing and the charging of calls that are switched by the SSP's. SSP's and SCP's are connected via a fast signaling network - SS7.

The SSP is usually a commercial telephone exchange with a modified call controlling software; the SCP contains the services' control logic and it has an access to the service database. In the IN, the calls' traffic flows through SSP's. The SCP makes part of the decisions that concern the routing and the charging of calls. During an IN call there may be one or more INAP dialogues between the SSP and the SCP. Each dialogue starts with an "initial detection point" (initial-dp) message.

Figure 5 SCP and SSP are connected via a SS7 signaling network. To communicate, SSP and SCP use the IN Application Part protocol.

Two examples of a service are the “free-phone”, and the “premium-rate phone”. A call to a free-phone number costs nothing to the caller; everything is paid by the called party that subscribes to the service. A call to a premium-rate number (for example a lawyer’s office) costs a lot to the caller; the money, after a deduction of the subscription fee goes to the bank account of the called party that subscribes to the service. IN offers a centralized implementation of those services. During the establishment of the call, SSP sends the dialed number and other data to the SCP, and the SCP returns the destination and the charging method for that call. Besides the dialed number, the destination and the charging of the call may depend on other things; for example, they may depend on the location of the caller and on the calendar time. During the same call several services can be executed sequentially, one after the other. “Nesting” is another way to combine services; a nested service starts and ends during an execution of another service.

IN is a network with a star topology¹. In a star network the nodes are of two kinds: central and peripheral. Peripheral nodes generate traffic that flows to the central node ; the central node is the computational bottleneck of the network. When IN contains more than one SCP, its architecture is a superposition of several star networks that share their peripheral nodes – SSP’s (see figure 6).

¹ Besides IN, amultitude of other networks have a star topology. Three of the many examples are: a satellite that switches traffic generated by earth stations, a controller that processes data gathered by remote sensors, and a GSM Mobile Switching Center with its Base Stations.

Figure 6 a) a star shaped network with three peripheral nodes; ‘‘P’’ denotes a peripheral node and ‘‘C’’ the central node. b) in the simplest case there is only one central node and one peripheral node. c) two star networks that share their peripheral nodes.

Conceptually, IN can be divided into three parts. The first part contains the SSP’s and other nodes that switch the calls traffic. In the second part are the services that the network offers. In IN, both the knowledge on what the services do, and the knowledge on how to access the data that the services need are located in a small number of central nodes (SCP’s). The third part is the SCP-SSP communication protocol that is the language in which the machines ‘‘talk’’. IN makes it easy to introduce and to modify services, but the use of any service has to be expressed as a sequence of INAP messages. In this sense the protocol defines the ‘‘intelligence’’ of the network.

2.2 Distributed overload control

When the traffic in the network is high, the SCP can get overloaded. IN has a distributed load control mechanism that uses ‘‘call gapping’’ method to limit the flow of messages towards the SCP. SCP detects the overload condition and the SSP’s filter the excessive traffic. When the load control in the SSP is active, SSP rejects, or ‘‘gaps’’ a certain part of calls before they start a dialogue. SSP does not abort dialogues in the middle².

Suppose that we have an Intelligent Network with one SCP and two SSP’s (see figure 7). The SCP contains a hierarchy of subsystems; each subsystem is located behind its controlling gate. All communication with the subsystem has to pass through the gate. The gate gathers statistics about the traffic, the condition of the subsystem, and the condition of the other parts of the SCP. From this data the gate computes the load level of the subsystem.

The normal load level of the subsystem is zero. When the load level changes from zero to one, the gate with a cooperation from SSP’s, will try to limit the message traffic that is directed towards the subsystem. To limit the traffic, the gate sends a call gapping request (cg-request) to both SSP’s. The message contains three groups of parameters (the following definitions are from section 7.3.6.1.1 of [4]). The first group is ‘‘gap criteria’’, the second group is ‘‘gap indicators’’ and the third group is ‘‘gap treatment’’.

Gap criteria identifies the portion of the traffic that is directed to the overloaded subsystem. Gap indicators define the maximum initial-dp’s rate, u and the duration of the restriction³;

² SSP can abort a dialogue in the middle, but not because of call gapping; for example, the reason can be that the SCP has failed to respond within a predefined period of time.

³ cg-request contains the minimum interval between two consecutive initial-dp messages, I . The maximum initial-dp’s rate, u is the inverse of this interval: $u = 1/I$.

between the arrival of the cg-request and the end of the duration, the rate should be at most this maximum (see figure 8) Gap treatment defines what to do with rejected calls. For example, the speech channel of a rejected call can be connected to a certain voice announcement, or to a busy tone. In addition, the cg-request message contains the “gap control” field. Gap control is either “SCP overload”, or “manually initiated”. When cg-request is due to the automatic load protection mechanism in the SCP, the value of the gap control field is “SCP overload”. When cg-request results from a command of the SCP’s operator, the value of the gap control is “manually initiated”.

Figure 7 IN with one SCP and two SSP's. Each subsystem in the SCP is located behind its controlling gate.

Manually initiated cg-requests have higher priority than messages with the same gap criteria and “SCP overload” gap control field ([4] section 7.3.6.3.1). After the manually initiated message has arrived, and before the duration specified in it has expired, SSP ignores non manual cg-requests that have the same gap criteria. This allows SCP’s operator to second-guess the machine.

When a cg-request arrives at SSP, SSP creates an “image” of the sending gate. The image works like a throttle-valve. With the aid of the gap criteria, the image identifies the portion of the traffic that is directed towards the overloaded subsystem, and limits the rate of this traffic stream. When the duration specified in the cg-request expires, SSP destroys the image (see figure 9).

Figure 8 “call gapping” is a rate dependent method for filtering traffic. When the traffic rate offered by the network is less than the bound rate u , there is no filtering. When the traffic rate exceeds u , SSP rejects part of the calls, so that the traffic rate after filtering is u . The characteristic of the real gapper is a smoothed approximation of the piecewise-linear ideal.

The gate in the SCP is “static”; it exists all the time. The image of the gate in the SSP is “temporary”; SSP creates the image when it receives a cg-request and it destroys the image when the duration specified in the cg-request has expired. When the SSP receives a cg-request with gap criteria that is the same as the criteria of an already existing image, the gap indicators, the gap treatment, and the gap control of that image will be updated.

Another approach is to view the images in the SSP as static objects with two states: “active” and “passive”. When the image receives a cg-request it turns active and starts to filter traffic. When the image is in the active state, it can receive several cg-requests from the gate in the SCP. Before the first cg-request has arrived, and after the duration specified in the last cg-request has expired the image is in the passive state.

When two of the subsystems in the SCP are overloaded at the same time, there are two images of their corresponding gates in the SSP’s. As more and more subsystems become overloaded, the logical structure of the images in the SCP’s starts to resemble the hierarchy of the SCP’s gates (see figure 10).

2.3 The synchronization problem

When one of the SSP’s starts a dialogue by sending an initial-dp message, the SCP has to decide if to send or not to send a new cg-request to that SSP. There shouldn’t be a new cg-request if there is an image of the overloaded subsystem’s gate in the SSP and if that image filters calls with the right set of parameters.

Figure 9 Subsystem C in the SCP is overloaded. By sending cg-requests, SCP has created images of C's gate in both SSP's. The images focus on the traffic that is directed towards the overloaded subsystem, and reduce that traffic by rejecting part of the calls. When the duration specified in the cg-request expires, SSP's will destroy the images.

Cg-request is a one-way message; there is no acknowledge message to it. The lack of feedback, together with the dynamic creation and destruction of call gapping entities in the SSP's makes the synchronization of the load protection mechanism difficult; SCP is "in the dark" as to what is the situation of the images in the SSP's.

Figure 10 As more and more subsystems become overloaded, the logical structure of the images in the SCP's starts to resemble the hierarchy of the SCP's gates.

2.4 How the problem was solved in Tele's IN product

In 1989 Telecom Finland (Tele) has contracted NTC and ICL for developing SSP and SCP that were based on BELLCORE's AIN 1 set of standards ([5], [6]). In this product the problem was solved by repeating the same cg-request after every initial-dp message from an SSP. This solution causes frequent updates of the images in the SSP's. It also adds to the traffic load on the SS7 signaling link that connects SSP and SCP.

3 RANDOMIZED UPDATES WITH PROBABILITY p

An improvement over the previous solution is as follows. Instead of sending a filtering request after every initial-dp message, the gate that controls the traffic into an overloaded subsystem, will send a filtering request in response to a certain proportion, p of the initial-dp messages. A gate can have several images that are located in different SSP's, and to distribute the filtering requests between them evenly, we employ a random number generator. An initial-dp will trigger a filtering request, if the output from a random number generator is at most p . Let's denote by "rand" a function that returns a uniformly distributed random variable in the range $[0, 1]$. For every initial-dp message, the decision if to send or not to send a cg-request is done as follows:

$$\text{send cg-request} := \text{rand} < p, \quad (1)$$

$$\text{where rand, } p \in [0,1];$$

The reduced amount of update messages comes at a price of increased interupdate time. Let's denote the rate of initial-dp's from an SSP towards the gate by r [initial-dp's/s], and the average inter-update time, i.e. the time between consecutive cg-requests that the gate sends to the SSP, by d . Clearly

$$d = 1/(rp) \quad (2)$$

Our goal is to find out how to choose p , but before that we have to answer the following question: What are the constraints on the value of d ?

To answer that question remember how a cg-request affects the SSP: if $r \leq u$, then the request will have little or no effect on it; if $r > u$, then r will be reduced to u for the duration specified in the request. It follows that the gate does not have to synchronize a SSP whose rate does not exceed u , and so we can restate the question in the following way: What are the acceptable values of d , if $r > u$?

This is an easy question to answer. Since r is bounded from below by u , d is bounded from above by $1/(up)$

$$d < 1/(up) \quad (3)$$

To choose p , we take a sender with rate that equals the upper bound, u as a yardstick, and we decide on the average update time D for that sender. Then

$$p = 1/(uD) \quad (4)$$

If possible, the value of D should be an order of magnitude higher than $1/u$, and it should be lower than the gap duration:

$$\frac{10}{u} < D < \text{gap duration}. \quad (5)$$

The response time to the sender with rate r is $1/(rp)$, or $(u/r)D$ seconds. The higher the r/u ratio, the faster will the cg-request be sent.

For example, if $u = 10$ [initial-dp's/s], gap duration equals 24 seconds, and the chosen value of D is 2 seconds, then $p = 1/20$. To keep the SSP's up to date with a 2 seconds' accuracy, we need to respond with a cg-request to only 5 percent of the initial-dp's.

Note that if the network is faulty, and if the probability of loosing any message is l , then the actual rate of any traffic source shrinks by the factor of $(1 - l)$ compared to its nominal rate, and the interupdate time grows by the factor of $1/(1 - l)$.

In the rest of this section we will consider the case where p is a periodic function of time, and the case where p depends on the rate of the traffic that flows to the overloaded subsystem.

3.1 Periodic p

$p(t)$ is a periodic function of time with a constant period T , that is at most the gap duration, and with a mean value, $E[p(t)]$ that is at least $1/(uD)$.

$$\begin{aligned} p(t) &= (t + T), \text{ where } T \leq \text{gap duration;} \\ E[p(t)] &= 1/(uD); \end{aligned} \quad (6)$$

Figure 11 an example of the periodic p 's shape.

For example, figure 11 shows a "square wave" p . Each period, T is divided into two parts. In the first part, $p = a$, and in the second part, $p = b$. The division into parts is defined by the value of time modulo T . In the first part, the time modulo T is between zero and T_1 ; in the second part, it is in the (T_1, T) interval.

$$\begin{aligned} p(t) &= \begin{cases} a & \text{if } 0 \leq (t \bmod T) \leq T_1, \\ b & \text{otherwise;} \end{cases} \\ E[p(t)] &= \frac{T_1(a - b)}{T} + b \end{aligned} \quad (7)$$

In particular, when $a = 1$ and $b = 0$, the behavior of the gate is completely deterministic. The gate alternates between sending a cg-request after every initial-dp for T_1 seconds, and not sen-

ding any cg-requests until the next period starts. The constraint $E[p(t)] \geq 1/(uD)$ translates into the following condition on the ratio of T_1 and T

$$\frac{T_1}{T} \geq \frac{1}{uD} \quad (8)$$

A response to an excessive traffic rate from a source will come faster when p is near its peak. It is to advantage to choose the period of $p(t)$ so, that it will fit into the gap duration an integral number of times: gap duration = kT , $k = 1, 2, \dots$, because then the end of the gap durations is likely to coincide with the time when $p(t)$ is at its peak.

We could also make p a function of the total traffic rate. For example, the gate will start to send cg-requests only when the total rate exceeds a certain threshold.

4 BROADCASTING

This solution is based on broadcasting the cg-request to every SSP and every SSP replying with an acknowledge message after it has either created or updated an image of the sending gate. The SCP keeps track of the acknowledgements, and after a time-out, retransmits the cg-requests to the SSP's that haven't sent an acknowledgement message. During the overload condition broadcasting is repeated periodically; the period is smaller or equal to the gap duration.

There are two problems with this solution. The first problem is that a reliable broadcasting is hard to implement. The second problem is that a fault can deactivate an image after an acknowledge has been sent.

5 RANDOMIZED BROADCASTING

The hard part of broadcasting is keeping track of the acknowledgement messages and resending the broadcasted message if necessary. If we omit that part, broadcasting is not as reliable, but much simpler to implement.

In a looseless network, that is, in a network with faultless nodes that always delivers every message to its destination, the gate has to broadcast only either when its parameters change, or when the gap duration expires. Suppose that the network is not looseless, and that the broadcasting is implemented without the acknowledgement part. To compensate for an occasional loss of some cg-requests the gate could broadcast more often, for example four times per gap duration.

Another possibility is that the gate will rebroadcast randomly, in proportion p of the incoming initial-dp messages. Each broadcast consists from n cg-requests, where n is the amount of SCP's in the network. To choose p we divide the right hand side of equation (4) in section 3 by n :

$$p = \frac{1}{nuD} \quad (9)$$

Here, D is the average interupdate time when the combined traffic rate from all SSP's is exactly nu .

For example, if $n = 5$, $u = 10$ [initial-dp's/s], gap duration equals 24 seconds, and the chosen value of D is 2 seconds, then $p = 1/100$.

6 THE STAMPING SYNCHRONIZATION METHOD

SCP maintains an integer that is called a "global stamp". Initially the value of the stamp is zero. When the parameters of any gapping gate in the SCP change, that gate increments the global stamp, reads its value, and stores it in its internal data. The stored stamp's value is called a "local stamp". The parameters of a gate change either due to an update, or because the subsystem behind the gate moves from one overload level to another. At any moment of time, the local stamp of a gate uniquely identifies that gate's current set of parameters.

```
function get stamp: integer                                     (10)
stamp := stamp + 1;
return stamp ;
endfunction;
```

Instead of being incremented with each update, the global stamp can be decremented, instead of holding an arbitrary integer the stamp can hold the current value of the SCP's computer clock. The important thing is that every time a gate needs a new stamp, it will get a unique value from a totally ordered set.

When sending a cg-request the gate will add the current value of its local stamp to it; when the SSP receives a cg-request, the message will contain the stamp of the sending gate. SSP copies that value from the cg-request to the data of the corresponding image. If a call has passed through the call gapping control of an image, the initial-dp message that is sent to the SCP will contain the stamp of that image.

The gate decides if to send or not to send a cg-request as follows. The initial-dp message can come with or without a stamp. If there is no stamp, there is no image of the gate in the SSP and the cg-request is sent. If there is a stamp, its value is either equal or different to the value of the local stamp. If the values are equal, the image of the gate is up to date; if the values are different, the gate sends a cg-request to the SSP.

```
send cg-request :=      ((no stamp in the initial-dp message) OR      (11)
                        initial-dp.stamp  $\neq$  local stamp of the gate));
```

6.1 The SCP's subsystems hierarchy

ETSI standard defines four kinds of gap criteria ([4] section 7.3.6.1.1.) The criteria are: "called address", "service key", "called address and service key", and "calling address and service

key”. Called address corresponds to the dialed number. Service key identifies the IN service that is needed by the call. Calling address corresponds to the telephone directory number of the calling subscriber.

In subsection 2.2 I have described the internals of the SCP as a hierarchy of subsystems that are protected by gates. A “subsystem” is a part of the SCP that has its own unique gap criteria. A traffic stream that matches gap criteria x , can be a part of a larger stream that matches criteria y ; in this case y is a subset of x . The less precise the criteria, the more of the traffic it matches, and the higher in the hierarchy is the corresponding subsystem. For example, a sequence of calls to numbers that start with the digits “3146” is a part of a sequence of calls to numbers that start with the digit “3”. It is that “part of” relation that imposes a partial order on the set of subsystems. A subsystem with criteria “called address = 3” is higher in the hierarchy than a subsystem with criteria “called address = 3146”. This view of SCP’s internals is what an external observer may deduce on the basis of SCP’s filtering requests.

Our model of the SCP is a Directed Acyclic Graph (DAG), where the vertices are subsystems, and where there is a path from a subsystem “A” to a subsystem “B” if the traffic that flows to “A” includes the traffic that flows to “B”. In DAG, the length of any path is bounded. In general, if there is a link from a subsystem “A” to subsystem “B”, a transaction that executes at “A” either will end at “A”, or it can continue to execute at “B”.

6.2 Filtering calls that match the criteria of several images

When the SSP contains two or more active images of gates that lie on the same path we have to decide how to filter calls that are directed to the subsystems behind those gates. The question can be restated as follows. How to filter a call whose data matches the gap criteria of more than one image?

Example: SSP contains two images with gap criteria “called address”. In the first image the criteria is “800”, and the bound rate is 10 [initial-dp’s/s]. In the second image the criteria is “80012”, and the bound rate is 5 [initial-dp’s/s] (see figure 12). When the subscriber dials “800123” we can filter the call either by the parameters of the first image, or by the parameters of the second image, or by the parameters of both images. Filtering by the parameters of the first image is wrong because then the subsystem behind the gate that has created the second image can get 10 [initial-dp’s/s] instead of the specified 5 [initial-dp’s/s]. Filtering by the parameters of the second image works in this case but it would be wrong if we would exchange the bound rates of the images. Filtering by the parameters of both images means that the call has to pass through the gap control of both, and if any of the images requires to gap the call, the call will be rejected. This is correct in all cases.

Note: according to the ETSI standard ([4] section 7.3.6.3.1), in the above example we should filter by the parameters of the second image.

Conclusion: When the data of a call matches the gap criteria of several active images, that call has to pass through the call gapping control of all those images, and if any image requires to gap the call, that call will be rejected.

Figure 12 SSP contains two images. The first image filters calls with dialed number that starts with ``800..'', and the second image filters calls with dialed number that starts with ``8001..''. As long as those two images coexist, according to the parameters of which image should we filter calls with dialed number ``8001..''?

This rule has the following effect on the stamping synchronization mechanism. Instead of zero or one stamp, the initial-dp message has an array of zero or more stamps. The array contains the current stamp values of all images that the call has passed. Before a transaction starts to execute in a subsystem, the gate of that subsystem checks if the value of its local stamp is in the array; when the value is absent, the gate will send a cg-request to the SSP.

The length of the stamps array is bounded by the depth of the subsystems' hierarchy in the SCP. The processing capacity of the SSP also sets a limit on the length of the array. We can't afford to have a large amount of dependent images in the SSP; when a call has to pass through many images, the computational overhead is high. I assume that in practice the array will have no more than, say, thirty stamps.

6.3 Rollover of the stamp value

Because the physical size of the global stamp variable is limited, the value of the stamp can not be incremented forever. For example, if the size of the variable is one octet, its maximum value is $2^8 - 1$, or 255; if the size of the variable is two octets, its maximum value is $2^{16} - 1$ that is about sixty five thousands. When a gate will try to increment a stamp that is at its maximum, the value will not increase; it will jump back to zero, or "roll-over".

After rollover, old stamp values are reused, and it is possible that two or more gates will have the same local stamp. The frequency of this event depends on the relative speed with which the gates update their parameters. We can either ignore the rollover, or we can deal with the problem in the following way: a rollover will cause every gate to get a new stamp value, even though there hasn't been any change in the gates' parameters. The resulting cg-requests will deliver the new stamps to the images in the SSP's.

6.4 The effect of network transmission delay

During the time that the cg-request with a new stamp is on its way to the SSP, the SSP may send a number of initial-dp messages with the old stamp value. Each of those messages will cause another cg-request (see figure 13). The transmission delay depends on the amount of traffic. In Telecom Finland's IN implementation, the measured delay is between 0.1 and 0.2 seconds. The transmission time is composed from the time it takes to transmit the message between the machines, and the time it takes a message to propagate through the protocol stacks of the sender and the receiver.

For example, when the transmission delay is 0.2s, the time between sending a cg-request and receiving a first initial-dp with the same stamp is at least 0.4s. If the traffic intensity is 10 [initial-dp's/s], there will be four repeated cg-requests during that time.

Figure 13 due to the network transmission delay, a cg-request may be repeated a few times

7 A COMBINATION OF THE STAMPING AND THE RANDOMIZED METHODS

We use the stamping method, but the gate will examine the stamps only in a certain proportion, p of the initial-dp messages. In section 3 I have shown that for a source with an average rate that exceeds u [initial-dp's/s], this guarantees a cg-request within $1/(pu)$ seconds on the average.

The advantage over a purely randomized update is that the gate doesn't send unnecessary cg-requests. If one of the stamps in the initial-dp message is equal to the local stamp of the gate, there is an image in the SSP that filters traffic with the right set of parameters. The advantage over pure stamping is that the stamp checking overhead is reduced by a factor of $1/(1-p)$.

8 CONCLUDING REMARKS

I have described five ways to solve the problem of overload protection synchronization in the Intelligent Network. Though formulated in terms of IN, the methods apply to any similar network, that is, a network with a star topology, traffic throttles in the peripheral nodes, and a bottleneck in the central node.

When the SCP synchronizes by broadcasting, it initiates the filtering requests by itself. The other methods synchronize in response to the incoming traffic. Thus we can say that broadcasting is an “active” synchronization method, and that the other methods are “reactive”. The intuition behind reactive synchronization is that the peripheral node (SSP) that loads the central node (SCP) the most, sends the highest proportion of messages to it, and thus it has the highest chance of receiving a filtering request.

The randomized reply, and the randomized broadcasting methods affect only the SCP, there is no need to modify the communication protocol between the SSP and the SCP. The other methods require small additions to the protocol.

For the broadcasting method, the addition is an acknowledgement to the cg-request. SCP should be able to broadcast a cg-request to all of the SSP's in the network, and to keep track of their acknowledgements. The problem with broadcasting is that a fault can deactivate the traffic throttle in the SSP after an acknowledgement message has been sent.

For the stamping method, there are two additions. The first addition is a stamp field in the cg-request message. The second addition is an array of zero or more stamps in the initial-dp message. The maximum length of the array is bounded by the depth of the subsystems' hierarchy in the SCP. This method is robust, it will work correctly in most circumstances.

For the randomized stamping method, the additions are the same as for the pure stamping. The advantage over pure randomized update is that there are less cg-requests. The advantage over pure stamping is that the stamp checking overhead is reduced.

Randomized reply is the easiest method to implement, and broadcasting is the hardest. Stamping and the randomized version of it are both reliable and not hard to implement. They are my methods of choice.

The discussion about the “answers vs. questions” throughput curve in section 1 is based on Kleinrock's paper [7]. The “gates and images” terminology is my own. The assumptions about the internals of the SCP, such as those made in subsections 2.2, and 6.1, are not restrictive; any SCP can be modeled in this way.

9 ACKNOWLEDGEMENTS

I'd like to thank Heikki Hämmäinen, Pekka Lahtinen and Richard Fehlman from Nokia Research Center for their comments, and also my colleagues in Nokia Telecommunications who have reviewed this text. Special thanks to Brian Weitch from HUT's Ship Laboratory for his encouraging remarks.

10 REFERENCES

- [1] Kumar A., Adaptive load control of the central processor in a distributed system with a star topology, *IEEE Transactions on Computers*, Vol. 38, No. 11, November 1989, pp. 1502 - 1512.
- [2] Lamport L., Time, clocks, and the ordering of events in a distributed system, *CACM* 21, 7 (July 1978) pp. 558--565.
- [3] Jefferson D., Virtual Time, *ACM transactions on Programming Languages and Systems*, Vol. 7, No. 3, July 1985, pp. 404--425.
- [4] ETSI IN CS1 INAP Part 1: Protocol Specification, Draft prETS 300 374-1, November 1993.
- [5] BELLCORE, AIN Release 1, Switching Systems Generic Requirements, SR-NWT-001123, May 1991.
- [6] BELLCORE, AIN Release 1, Switching Systems Report, SR-NWT-001754, September 1990.
- [7] Kleinrock L., On the modeling and analysis of computer networks, *Proceedings of IEEE*, Vol. 81, August 1993, pp 1179-1191.

11 BIOGRAPHY

Philip Ginzboorg received the B.S. degree in Electrical Engineering from Beer Sheva University, Israel, in 1989. From 1989 to 1995 he worked in Nokia Telecommunications on, among other things, line signaling, routing and overload control. Since 1995 he is a member of Communications Systems Laboratory in Nokia Research Center, Helsinki. His current work focuses on the control of services in broadband networks.