多业务环境中 SCP 过载控制研究 [1]

李彤红 廖建新 陈俊亮

(北京邮电大学程控交换技术与通信网国家重点实验室 北京 100876)

摘 要 本文首先对 SCP 的处理模型作了抽象，就多业务环境中过载检测算法和 SCP 的最大处理能力的定量分析进行了讨论。接着提出了一种基于动态调整的二级控制算法用于 SCP 的过载控制。理论分析和模拟都证明了此算法具有公平性好、效率高的优点。

关键词 智能网，多业务，过载检测，动态调整，二级控制算法

中图号 TN913.2

1 引 言

智能网 (IN) 的目的是为了快速而有效地提供满足不同用户要求的新业务。为了实现这个目的，在 IN 体系结构中，将交换和控制分离，使交换机 (SSP，业务交换点) 仅执行基本的交换功能。IN 业务需要的业务逻辑程序和用户数据分布在 SCP(业务控制点)，SCP 通过执行业务逻辑程序，控制业务的执行。业务逻辑程序由 SMS(业务管理系统) 下载到 SCP 中。通常，每个 SCP 中存在几个不同的业务逻辑程序，因而能够实现多种业务。

当同一时间到达 SCP 的业务请求非常之多超过 SCP 的处理能力时，SCP 将过载。如果 SCP 不对 SSP 进入 SCP 的业务请求数加以限制的话，成功处理的业务请求数 (即吞吐量) 将下降非常严重。关于 IN 过载算法，文献 [1–6] 对此作了研究，但这些算法都是在 SCP 执行单业务的情况下进行的，此时 SCP 处理每个业务的时间、SCP 的最大处理能力都是固定的，过载的检测和控制都很简单。在多业务环境中，由于每个业务进行不同的呼叫处理，执行不同的业务逻辑，因而处理时间是不同的。另外，在过载的 SCP 上同时执行多种不同的业务，因此业务的平均执行时间将根据不同的业务流而发生变化，SCP 单位时间能够处理的最大业务数不是固定的。

作为一个例子，下面说明 SCP 单位时间能够执行的业务数随着业务流的不同而不同。假设 SCP 能处理三种业务 A、B、C。每个业务的处理时间分别为 1、5、20ms。在业务流 X 下，业务 A 的业务请求率为 80 呼叫 /s，B 为 10 呼叫 /s，C 为 10 呼叫 /s。因此，总的输入率为 100 呼叫 /s。在这种情况下，SCP 的负载为 0.33，SCP 每秒可以处理 100 个呼叫。在业务流 Y 下，业务 A 的业务请求率为 10 呼叫 /s，B 为 10 呼叫 /s，C 为 80 呼叫 /s。此时，总的输入率也为 100 呼叫 /s，与业务流 X 相同。但是此时 SCP 的负载大于 1，显然 SCP 每秒不能处理 100 个呼叫。

SCP 过载控制算法必须满足下面要求：(1) 有效性。也就是说，在负载非常重的情况下，SCP 的吞吐量不会下降。在多业务环境下，SCP 的最大处理能力用什么来表示是一个值得讨论的问题。从上面的例子来看，如果按照 SCP 每秒可处理的最大呼叫数来表示的话，将是不确切的。因此必须重新定义新的指标。(2) 公平性。在多业务环境中，当 SCP 过载时，SCP 不能对没有产生过载的业务进行限制。业务运营者可根据自己的需要，如每种业务的处理时间和从每种业务得到的收益，规定在过载情况下，各种不同业务的处理比率。

[1] 1997-09-02 收到，1998-04-20 定稿

863 高技术项目、国家自然科学基金项目、邮电部重点项目、教委博士点基金项目资助

(3) 对不同的过载模块、不同的业务流、不同的用户行为 (用户重试) 和 SCP 结构都适用.

(4) 算法必须易于实现。

本文是这样组织的，第 2 节对 SCP 的处理模型作了抽象，得出了多业务环境中的过载检测算法和 SCP 最大处理能力的测量指标。第 3 节对 SCP 过载控制框架作了描述，并给出了过载控制算法 A，这是一种基于动态调整的二级控制算法。第 4 节证明了此算法具有公平性好、效率高的特点。第 5 节给出了仿真结果。

2 多业务环境中的过载检测算法、最大处理能力的测量指标

2.1 SCP 处理模型

对任何 SCP，不管怎样实现，都存在下面四种不同的用户进程：接收消息进程 (receive)、发送消息进程 (send)、主控进程 (main)、业务逻辑执行进程 (SP)。其中，receive 进程处理优先级最高。每次产生中断，即 SCP 收到来自 IN 其它节点 (如 SSP) 发来的消息时，receive 将相应的消息放入接收缓冲区内。main 是一个循环过程，它是按以下步骤执行的：

(1) 判断接收缓冲区是否为空，不空的话，将消息解码后放到消息队列中去。

(2) 从队列头取消息，判断消息在队列中的等待时间是否超过 T(假设智能用户的最大等待时间为 T，一般为 1 到 2s。当消息在队列中的等待时间超过 T 时，本次呼叫将失败，消息不再有效.)。如果超过 T，则将消息抛弃。然后根据消息的类型，转不同的分枝。如果消息是送往其它节点的，则转 (3); 如果是送往本节点的，则转 (4); 消息队列是空的话，转 (1)。

(3) 调用 send 进程，将消息编码后发到其它节点。send 执行完后，转 (1)。

(4) 调用业务逻辑执行进程 SP，SP 根据消息的类型调用相应的业务逻辑。业务逻辑在执行的过程中，产生不同的消息，放到消息队列。每个呼叫业务逻辑的执行可看作是一个有限自动机。它总是处于某个状态，等待某种消息。得到消息后，进行处理，并发送新的消息，然后进入一个新的状态。SP 执行完后，转 (1)。

2.2 多业务环境中的过载检测算法、最大处理能力的测量指标

每个进程执行时都要占用 CPU 的时间。当 SCP 负载小的时候，消息队列中消息个数不多，每个消息的等待时间也不长，因此不存在消息丢弃现象。随着业务请求数的增加，SP 进程占用 CPU 的时间 (称为 CPU 占用率)、消息队列中消息数、消息等待时间都将增加。当业务请求数增加到一定程度时，消息队列中的消息开始超时，SCP 开始丢弃消息，理论和模拟 [6] 都证明，SCP 如果不要求 SSP 对其进入 SCP 的呼叫数进行限制的话，在负载非常重的情况下，SCP 成功完成的呼叫数将严重下降。

从上面可以看出，SCP 是否过载可根据消息队列中消息是否超时来判断。SCP 每隔 5s (这个值可经过实验得到，综合考虑了处理开销和响应速度的关系) 检查一次，看这段时间内是否发生消息超时。若发生消息超时，则认为 SCP 已过载。SCP 的最大处理能力以前是根据 SCP 单位时间能够完成的最大呼叫数来量度的，但在多业务环境中，由于不同业务处理时间不同导致单位时间能够完成的最大呼叫数是变化的，因此 SCP 的最大处理能力不能根据最大呼叫数来量度。我们知道，SP 进程的 CPU 占用率 p 和 SCP 单位时间成功完成的呼叫数有以下关系：最开始时 SCP 单位时间成功完成的呼叫数是随着 p 的增加而增加的，但当 p 增加到某个值 p^* 以后，再增加 p，成功完成的呼叫数反而减小。在 p 达到 p^* 以前，SCP 没有丢弃消息，所有消息都能成功处理。当 p 超过 p^* 以后，SCP 开始丢弃消息。因此可将 p^* 作为 SCP 最大处理能力的指标值。SCP 在检测到过载时，要尽量使 SP 的 CPU 占用率达到此值。p^* 可通过测试来确认。在本文后面的讨论中，都假设此值是事先指定的，p^* 一般设置比测量值小一些。

测量 SP 进程 CPU 占用率的方法很多．最简单的一种是在测量时间 T 内，每隔一段很短的时间对 CPU 执行的指令抽样一次，看是否在执行 SP 进程，如果是，将计数器加一．T 时间到后，将计数器的值除以抽样次数，即得 CPU 占用率．显然，抽样次数越多，测量的值越准确．但考虑 CPU 的执行开销，抽样次数不宜太大．关于抽样次数的确定在统计学中已有成熟的理论，另外有些编程语言也提供了测量某个过程执行效率的库函数，因此本文不对此进行讨论．

3　过载控制算法

3．1 过载控制的框架

IN 过载控制的框架如图 1 所示 (为了简单，只画了一个 SSP，其它 SSP 类似．) 假设 SCP 可执行 n 种业务，SSP 的个数为 m 个，SSP 可向 SCP 请求 n 种业务．IN 业务需要的业务逻辑程序和用户数据分布在 SCP．智能呼叫是在 SSP 中检测的，SSP 给 SCP 发送信息流，请求 SCP 执行业务逻辑程序，控制业务的执行．

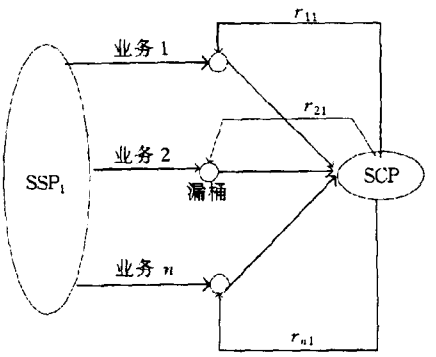

图 1

当 SCP 检测到过载后，将给 SSP 发信息流，请求 SSP 对引起过载的业务进行限制．SSP 只对每个呼叫送往 SCP 的第一条消息进行限制，一旦此呼叫的第一条消息通过了，将不对它以后的消息进行限制．文献 [7] 通过理论分析和模拟，得出了漏桶算法比 callgap 和 percent 漏桶算法稳健性强的结论，因此本文将在 SSP 中限制呼叫业务时使用漏桶算法，并将漏桶设置为理想漏桶 (将令牌缓冲区设置为大于 10 即可)．SCP 过载控制算法的关键是怎样确定漏桶的令牌产生速率 r_{ij}，使算法满足第 1 节提出的要求．

3．2 过载控制算法

从第 1 节，我们知道 SCP 过载控制算法必须满足四个要求：(1) 有效性．在 SCP 过载时，必须确保 SP 进程的 CPU 占用率达到 p^*．(2) 公平性．对那些引起过载的业务，其到达 SCP 的速率必须符合运营者事先规定的比例，对没有引起过载的业务不能进行限制．假设比例因子都是整数，第 i 个业务的比例因子为 $y_i(1 \le i \le n)$．(3) 对不同的过载模块、不同的业务流、不同的用户行为 (用户重试) 和 SCP 结构都适用．(4) 算法必须易于实现．

我们提供的算法 A 是一个基于动态调整的二级控制算法．它由业务速率更新算法 (Update) 和分配算法 (Distribute) 组成．算法 A 每隔一段时间 T(一般为 5s)，判断是否取消过载控制，如果不取消的话，则调用 Update 确定下一段时间 T 内每种业务总的接收速率．(Update 根据上一个时间 T 内 SCP 的测量值进行计算，测量参数将在下面说明．) 接着 SCP 调用 Distribute 确定每个 SSP 中每种发生过载业务相应漏桶的控制参数，使每种过载业务总的输出速率等于 Update 确定的值，Distribute 对没有过载的业务不进行限制．

在每个时间 T 内，SCP 需要测量下面参数：(1) 每种业务单位时间的到达率．(2)SP 进程的 CPU 占用率．(3) 每个 SSP 中每种业务的输入速率，这个速率 SCP 是测量不到的．SSP 可在收到算法 A 的控制命令时，每隔一秒种通过信息流向 SCP 报告前一秒内每种业务的到达速率，SCP 可通过此信息流计算出时间 T 内每个 SSP 每种业务的平均输入速率 (只是一个估计值)．为了说明的方便，我们用下面符号表示测量参数：$A_{ij}(k)$ 为第 k 个时间 T

内业务 i 到达 SSPj 的平均输入速率， $C_i(k)$ 为第 k 个时间 T 内业务 i 到达 SCP 的平均速率， $p(k)$ 为第 k 个时间 T 内 SP 进程的 CPU 占用率。

下面是算法 A 的描述。

算法 A

(1) 初始限制，给所有的 SSP 发信息流，信息流中参数包括应该限制的业务种类、控制时间 T_0 ，使所有 SSP 在 T_0 内停止向 SCP 发送业务请求。

(2) 置 $C_i(1)=C_i(1\le i\le n)$ 。调用 Distribute ，根据 Distribute 计算出来的值给所有的 SSP 发信息流，信息流中参数包括应该限制的业务种类、控制时间 (取为 T) 和令牌控制参数 r_{ij} 。

(3) 在 T 时间内，对各种业务进行监视，得到需要的测量值。

(4) T 时间到时判断 SCP 是否过载，过载的话，算法转 (1) 重新执行；不过载的话，转 (5) 。

(5) 判断是否取消控制，取消控制的话，则算法结束．不取消的话，转 (6) 。

(6) 调用 Update 确定下一段时间 T 内每种业务总的接收速率。

(7) 调用 Distribute 确定每个 SSP 中每种过载业务相应漏桶的控制参数，给所有的 SSP 发信息流，控制相应的漏桶； Distribute 对没有过载的业务不进行限制。

(8) 转 (3) 。

在算法的执行中，由于原来没有过载的业务其请求数可能发生显著的变化，因此应增加 (4) ，判断 SCP 是否过载，过载的话，应放弃算法已有的执行结果，使算法重新执行。

$C_i(1\le i\le n)$ 可以这样得到：先测量每种业务的平均执行时间 (假设第 i 种业务的平均执行时间为 μ_i) ，然后解方程 $\sum_{i=1}^{n}xy_i\mu_i=p^*$ ，设其解为 x^* 。 C_i 可定义为 y_ix^* 。

由于在检测到过载时， SCP 中的消息队列中积累了许多业务请求消息，因此应使所有 SSP 在 T_0 内停止向 SCP 发送业务请求，使 SCP 尽快将队列中的消息处理完。 T_0 可根据消息队长和消息的最长处理时间通过计算得到，一般取为 2s 。过载控制的取消可以这样判断：如果在第 k 个时间 T 内，对所有限制的业务 i 都有 $\sum_{j=1}^{m}A_{ij}(k)<C_i^*(k)$ ，并且 SCP 不过载。 $C_i^*(k)$ 的含义将在下面说明。

算法 Update

如果 $\sum_{j=1}^{m}A_{ij}(k)>C_i(k)$ ，则 $C_i^*(k+1)=(p^*/p(k))C_i(k)$ ；如果 $\sum_{j=1}^{m}A_{ij}(k)=C_i^*(k)$ ，则 $C_i^*(k+1)=[(1+\alpha)p^*/p(k)]C_i(k)$ 。

$C_i^*(k)$ 为第 k 个时间 T 内业务 i 总的接收速率。 α 是为了防止对没有引起过载的业务进行限制，有关它的作用将在第 4 节进行解释。 α 可取为 0.1 。

算法 Distribute

如果 $\sum_{j=1}^{m}A_{ij}(k)\le C_i^*(k+1)$ ，则在第 $k+1$ 个时间 T 内不对业务 i 进行限制；如果 $\sum_{j=1}^{m}A_{ij}(k)>C_i^*(k+1)$ ，则按下面步骤计算第 $k+1$ 个时间 T 内 SSP_j 中业务 i 的漏桶控制参数 r_{ij} 。

(1) 计算 $\mathrm{avg}=C_i^*(k+1)/m$ 。

(2) 计算 $B_1=(C_i^*(k+1)-\sum_{j\in L}A_{ij}(k))/l$ 。 $L=\{j,A_{ij}(k)\le \mathrm{avg}\}$ ， l 为集合 L 的大小。

(3) 计算 $r=(C_i^*(k+1)-\sum_{j\in G}A_{ij}(n))/g$ 。 $G=\{j,A_{ij}(n)\le B_1\}, G'=\{j,A_{ij}(n)<B_1\}$, g 为集合 G' 的大小。 $r_{ij}=r$ ，当 $j\in G'$ 。

SCP 将通过信息流对 $SSP_j(j \in G')$ 中的业务 i 进行限制，其控制参数为 r_{ij}；对 $SSP_j(j \notin G')$ 中的业务 i 不进行限制。

4 算法 A 公平性和有效性的证明

假设各种业务的平均到达率分别为 $A_1, A_2, \cdots, A_n$，并且 $C_i < A_i$，则所有业务都发生过载。从算法 A 可知，$C_i(1) = C_i(1 \le i \le n), p(1) = C_1\mu_1 \cdots + C_n\mu_n$。由于 $C_i = y_i x^*$，所以 $p(1) = y_1 x^* \mu_1 + \cdots + y_n x^* \mu_n = p^*$。显然 SCP 能到达最大处理能力 p^*，SCP 单位时间每种业务的处理比率也符合运营者事先的规定，即满足公平性假设。

假设 SCP 过载，但并不是所有的业务都满足 $C_i < A_i$。定义集合 $J_1 = \{i, C_i \ge A_i\}$，集合 $J_2 = \{i, C_i < A_i\}$。

因为 $C_i(1) = A_i$，当 $i \in J_1$; $C_i(1) = C_i$，当 $i \in J_2$; $p(1) = \sum_{i \in J_1} A_i\mu_i + \sum_{i \in J_2} C_i\mu_i < p^*$; 所以 $C_i^*(2) = p^* A_i / p(1)$(为了证明的方便，我们假设 $\alpha = 0$)，当 $i \in J_1$；$C_i^*(2) = p^* C_i / p(1)$，当 $i \in J_2$; 所以 $C_i(2) = A_i$，当 $i \in J_1$; $C_i(2) = p^* C_i / p(1)$, 当 $i \in J_2$; 所以 $p(2) = \sum_{i \in J_1} A_i\mu_i + \sum_{i \in J_2} C_i(2)\mu_i < p^*$。这就是说，$p(2)$ 仍没有达到 p^*。但我们可以证明，当 k 增大时，$p(k)$ 将收敛到 p^*。下面给出其证明。

因为 $p(k+1) = \sum_{i \in J_1} A_i\mu_i + \sum_{i \in J_2} p^* C_i(k)\mu_i / p(k)$。又因为 $p(k) = \sum_{i \in J_1} A_i\mu_i + \sum_{i \in J_2} C_i(k)\mu_i$, 所以 $p(k+1) = \sum_{i \in J_1} A_i\mu_i + p^*[1 - (\sum_{i \in J_1} A_i\mu_i)/p(k)]$。

设 $\sum_{i \in J_1} A_i\mu_i = A$，所以 $p(k+1) = A + p^*(1 - A/p(k))$。

对上式进行递推，可得 $p(k) = 1/\{p^*/A)^{k-1}[1/(p(1) - p^*) - 1/(A - p^*)] + 1/(A - p^*)\} + p^*$。所以 $\lim_{k \to \infty} p(k) = p^*$　　证毕

因此，在不是所以业务都过载的情况下，算法 A 也满足公平性和有效性条件。

下面解释 α 参数设置的必要性。假设业务 i 没有过载，从上面的证明可知，当 k 很大时，$p(k)$ 可达到 p^*。但由于业务请求是随机变化的，因此当业务 i 的请求增加时 $p(k)$ 将大于 p^*。因为 $C_i^*(k+1) = p^* C_i(k)/p(k) < A_i$，显然如不设置 α 的话，将对业务 i 进行限制。设置 α 可允许没有过载的业务其呼叫请求数有一定的波动。

5 模拟结果

仿真模型是根据图 1 建立起来的，其中 SCP 的个数为 1，SSP 的个数为 3。SCP 是一个单服务排队系统，采用先来先服务机制 (对于同时到达的消息采用随机服务机制)，其服务时间根据不同的业务而不同。SCP 在处理消息时，先判断其在队列中的等待时间是否超过 1s，超过的话，则将消息抛弃，处理时间为 0.005s。SCP 消息队列的长度为 200，队列满时，到达的消息将自动抛弃，处理时间为 0。SCP 可处理两种业务，业务 1 的处理时间为 0.02s，业务 2 的处理时间为 0.01s。业务到达 SSP 服从泊松分布。漏桶中令牌缓冲区的大小为 4，p^* 为 0.8，T 和 T_0 都为 5s。在过载时，如果两种业务的到达率都很高，则要求业务 1 和业务 2 成功完成的呼叫数之比接近 1:2。

一般来说，处理一个 IN 业务需要在 SSP 和 SCP 之间来回交换几条消息。为了简单，本仿真模型假设 SSP 和 SCP 只来回交换一次消息，即先由 SSP 向 SCP 发业务请求，然后 SCP 给 SSP 返回请求结果。这样做对过载算法的结果没有影响，因为过载控制只限制初始业务请求消息，其它消息的处理时间可以增加到初始业务请求消息的处理时间上。

下面给出各种情况的结果。X 轴以 5s 为一个单位，Y 轴为 5s 内相应结果的平均值。

(1) SCP 过载时业务 1 和业务 2 的到达率都很高。其具体参数如下：在 40s 以前， SSP 中业务 1 和业务 2 的到达率都为 3.3，此时 SCP 没有过载。在 40 到 120s 之间 SCP 过载，每个 SSP 中业务 1 和业务 2 的到达率都为 17。在 120 到 200s 之间， SCP 不过载， SSP 中业务 1 和业务 2 的到达率都为 3.3。图 2 给出了业务 1 和业务 2 单位时间成功完成的业务数，虚线表示业务 1，实线表示业务 2。图 3 给出了 SP 进程的 CPU 占用率。从图 2 中可以看出，在过载时，业务 1 和业务 2 成功完成的呼叫数之比接近 1:2，满足要求的公平性条件。在 SCP 没有过载时，控制算法不对 SSP 的呼叫进行限制。从图 3 中可以看出，在过载时， SP 进程的 CPU 占用率在 0.8 左右，显然达到要求的 CPU 最大处理能力。

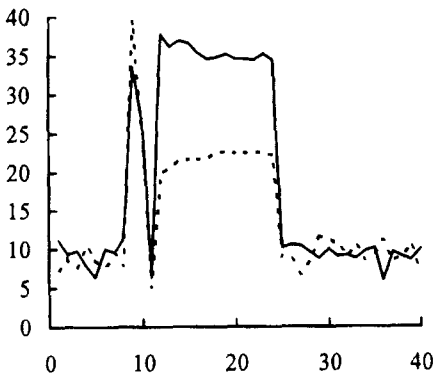

图 2

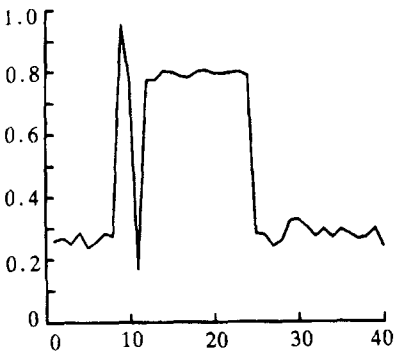

图 3

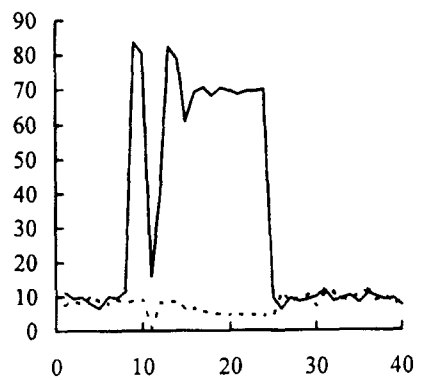

图 4

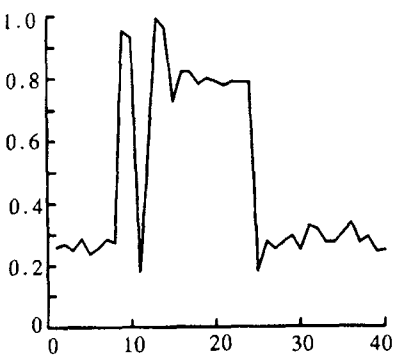

图 5

(2) SCP 过载时业务 1 的到达率很低，业务 2 的到达率很高。其具体参数如下：在 40s 以前， SSP 中业务 1 和业务 2 的到达率都为 3.3，此时 SCP 没有过载。在 40 到 120s 之间 SCP 过载，每个 SSP 中业务 1 的到达率仍为 3.3，业务 2 的到达率上升为 33。在 120 到 200s 之间， SCP 不过载， SSP 中业务 1 和业务 2 的到达率都为 3.3。图 4 给出了业务 1 和业务 2 单位时间成功完成的业务数，虚线表示业务 1，实线表示业务 2。图 5 给出了 SP 进程的 CPU 占用率。从图 4 中和图 5 可以看出，在过载时，不对业务 1 进行限制， SP 进程的 CPU 占用率在 0.8 左右，满足要求的公平性和有效性条件。

6 结　论

本文的分析工作、模拟工作以及所得到的结论对于智能网中 SSP 、 SCP 的合理设计和智能网可靠和有效的运行具有一定的实际意义。本文提出的过载检测算法和二级控制算法将用于北京邮电大学承担的移动智能网项目的过载控制中。

致谢 感谢北京邮电大学国家重点实验室智能网研究组全体工作人员提供的帮助！

参 考 文 献

[1] Smith D E. Ensuring robust call throughput and fairness for SCP overload controls. IEEE/ACM Trans on Networking, 1995, 3(5): 538–548.

[2] Phan X H, Betts R. Congestion control for intelligent networks. Computer Networks and ISDN Systems, 1994, 26(5): 511–524.

[3] Ryoichi Kawahara, Takuya Asaka, Shuichi Sumita. Overload control for the intelligent network and its analysis by simulation. IEICE TRANS.COMMUN., 1995, E78-B(4): 494–503.

[4] Northcote B S. Analysis SCP congestion control strategies: A study of control dynamics. in IEEE Intelligent Network Workshop IN'96, Melbourne, Australia: 1996.

[5] Kihl M, Nyberg C. A study of methods for protecting SCP from overload. IEE Telecommunications' Conference Publication no.404, 1995: 125–129.

[6] Farel R A, Gawande M. Design and analysis of overload control strategies for transcation network database. in Proc.13th Int.Teletraffic Congr., Copenhagen, Denmark: June, 1991, 115–120.

[7] Berger A W. Overload control using rate control throttle: selecting token bank capacity for robustness to arrival rates. IEEE Trans on Automatic Control, 1991, AC-36(2): 216–219.

THE STUDY OF SCP OVERLOAD CONTROL IN MULTI-SERVICE ENVIRONMENT

Li Tonghong　　Liao Jianxin　　Chen Junliang

(*National Laboratory of Switching Technology and Telecommunication Networks, Beijing University of Posts and Telecommunications, Beijing* 100876)

Abstract This paper first gives a SCP abstract model, then SCP's overload detection and maximum processing capability are discussed quantitatively. Based upon dynamic adjustment, a new two-level SCP overload control algorithm is proposed. Theoretical analysis and simulation prove the algorithm's effectiveness and fairness.

Key words IN (intelligent network), Multi-service, Overload detection, Dynamic adjustment, Two-level control algorithm

李彤红： 男，博士生，主要研究方向：移动智能网的设计、智能网的过载控制和性能分析、移动计算.

廖建新： 男， 1965 年生，副教授，主要研究方向： ATM 网络的性能分析、移动网和智能网的综合、宽带智能网的理论研究和性能分析.

陈俊亮： 男， 1933 年生，教授，中国工程院院士和中国科学院院士，主要研究方向： ATM 和宽带智能网.