Implementation and Analysis of PCC (Parallel Connection Control)

Malathi Veeraraghavan
Bell Laboratories
Room 4F-509
101 Crawfords Corner Road
Holmdel, NJ 07733
Email: mv@bell-labs.com

Gagan L. Choudhury
AT&T Laboratories
Room 1L-238
101 Crawfords Corner Road
Holmdel, NJ 07733
gagan@buckaroo.att.com

Madhu Kshirsagar
Bell Laboratories
Room 4F-513
101 Crawfords Corner Road
Holmdel, NJ 07733
mmk@dnrc.bell-labs.com

Abstract

In earlier papers, we proposed the PCC (Parallel Connection Control) algorithm for the setup and release of on-demand ATM (Asynchronous Transfer Mode) connections. This paper describes an implementation and analysis of PCC. The PCC algorithm is implemented on a testbed of three types of ATM switches from which service time measurements (software execution times) are obtained. These measured service times are used as input data for an analytical queueing network model to characterize the PCC end-to-end setup delay and maximum throughputs. Assuming all connections pass through ten switches, for this measured data, PCC has a theoretical maximum throughput of 343 calls/sec/switch, and at about 90% of this maximum limit (i.e., at 310 calls/sec/switch), the mean end-to-end connection setup delay is only 43ms. We also compare PCC performance against an equivalent sequential connection setup/release approach. We observe that PCC end-to-end setup delay is much smaller than the sequential setup delay (34% at low load and even smaller at higher load). Also, the per-switch maximum throughput of the PCC approach is 80% more compared to the sequential approach.

1 Introduction

ATM (Asynchronous Transfer Mode) networks are based on connection-oriented communication in that a connection is set up prior to data exchange and released after data exchange. The commonly accepted solution for ATM connection management is the ATM Forum PNNI (Private Network-Connect Interface) standard [1]. This standard consists of two protocols: the PNNI routing protocol and the PNNI signaling protocol. During connection setup, switches consult the data updated by the PNNI routing protocol to determine the "shortest-path" route for the connection. The first switch in each PNNI peer group on the path of the connection determines the "shortest-path" route of the connection through that peer group. Following this route computation, connection setup is performed at each switch within the peer group sequentially.

We observe that since the first switch in a peer group identifies the switches in that peer group through which the connection will be routed, connection setup at these switches can be accomplished in parallel. This will improve end-to-end connection setup delay. However, the first switch will have the added burden of sending messages to the switches on the selected route within its peer group, and coordinating responses. This will negatively impact the call handling capacity of switches which are already expected to be short on computing resources. To address this drawback, we propose separating the route computation function from other connection management functions performed in switches (such as switch resource management), and locating the route computation function in new servers called connection servers distinct from switches. This improves per-switch call handling capacity by reducing the number of functions performed by the switch processors. These two concepts of parallelizing connection setup operations at switches, and performing route computation in separate servers, constitute the PCC (Parallel Connection Control) algorithm, proposed in [2, 3]. Using this scheme, connection setup actions at the switches on the computed route within each peer group are executed in parallel. Connection setup proceeds sequentially from one peer group to the next on the route of the connection.

Section 2 gives a brief overview of the PCC approach. The PCC implementation is described in Section 3, and service time measurements obtained from this implementation are listed in Section 4. Using the measured data, a comparative analysis of the PNNI signaling solution and the PCC solution is carried out as described in Section 5. Conclusions drawn from this analysis are provided in Section 6.

2 Parallel Connection Control (PCC)

In the PCC solution, connection control functionality is distributed between two entities, connection servers and switch resource servers, as shown in Fig. 1. A switch resource server is associated with each switch. Multiple connection servers are present in each peer group of a PNNI-standards-based network. The PNNI routing proto-
connection at switches send topology state packets to connection servers rather than to each other. This removes the software needed to receive and parse PNNI routing protocol topology state packets from the switch processors. In addition, UNI (User-Network Interface) [4] and PNNI signaling, and route computation modules are also moved out of the switch processors to connection servers.

The bidirectional two-party PCC connection setup procedure is summarized here. Readers are referred to [2, 3] for details. Upon receiving a Setup-connections message, the route computation module of the connection server (Fig. 1) determines the route of the connection using the topology data collected by the PNNI routing protocol. This computation is similar to the DTL (Designated Transit List) computation in PNNI networks. The connection server then establishes the first segment of the connection through switches in its peer group by executing switch setup actions in parallel using a two-phase message exchange. In the first phase, it requests the switch resource servers of the switches on the route within its peer group to reserve resources for the connection using the Reserve-resources messages (one message per switch). This phase is executed in parallel at these switches.

The switch resource management module of each switch resource server (Fig. 1) receives this message and performs two steps:

- Step 1: CAC (Connection Admission Control) procedures to determine if the new connection can be admitted without disrupting any QoS (Quality-of-Service) guarantees given to pre-existing connections.
- Step 2: Selection of VPI/VCIs (Virtual Path Identifiers/Virtual Channel Identifiers) on the ports to the neighbor switches. The selected channel identifiers are included as parameters in the responses.

If this first phase is successful, the connection server initiates the second phase in which the switch resource servers configure the switches by sending Configure-switch messages (one message per switch). This phase is also executed in parallel at the switches on the route in the peer group. Each switch resource server performs the next two steps of connection setup:

- Step 3: Port/VPI/VCI translation table entries are made for user plane ATM data cell routing;
- Step 4: Parameters for runtime algorithms such as scheduling, priority control, and usage parameter control, are set based on the QoS requirements.

Upon successful completion of this phase, the segment through the first peer group is fully established. The reason we use a two-phase approach is that channels selected by each switch in the first phase (Step 2) are needed to configure switches in the second phase (Step 3).

If the connection spans multiple peer groups, the connection server sends a PNNI signaling message [1] to a connection server in the next peer group to request the setup of the next segment of the connection. Upon completion of setup through the last peer group, the connection server in this peer group sends an Offer-connections message to the far end host. If accepted, data transmission is initiated by the connection server sending a Start-traffic message. Response to the segment setup PNNI signaling messages traverse backwards through connection servers until the first connection server is reached. This entity then sends a Start-traffic message to its end host to initiate data transmission from the first end host.

UNI (User Network Interface) [4] signaling messages, can be used for communication between end hosts and connection servers. For example, the UNI SETUP message can be used instead of the Setup-connections message.

3 Implementation

In this section, we describe an implementation of the PCC algorithm. The ATM testbed network available in our research laboratory is shown in Fig. 2. It consists of two

- MINT switches [5] and two Fore Systems switches, and three VP switches. There are two types of multimedia end hosts, those connected to Fore switches and those con-
nected to MINT switches. The end hosts connected to the Fore switches are SUN workstations with a Fore ATM interface adapter card, and a JPEG video card. The audio capability is built-in with the workstation. The end hosts connected to the MINT switches are SUN workstations connected via ethernet to terminal adapters. Each terminal adapter has a control processor, executing the VxWorks operating system, an ATM network interface card, an audio card, and an NTSC video card. Details about the MINT terminal adapter can be obtained from [6].

The PCC software consists of three distinct types of UNIXTM processes: connection server, switch resource server, and end host signaling process. The connection server is shown as an explicit "box" in Fig. 2 to illustrate that it is run on a separate SUN workstation. The switch resource servers are collocated with ATM switches. Three types of switch resource servers are implemented, one for each type of ATM switch. One end host signaling process is executed per end host.

The different processes communicate with each other using IP over ATM or IP over ethernet. The PCC messages shown in Fig. 2 are transported in TCAP (Transaction Capabilities Application Part) transactions, where TCAP is part of the SS7 (Signaling System No. 7) protocol for invoking remote operations [7]. We transport TCAP messages as UDP/IP (User Datagram Protocol/Internet Protocol) [8] packets instead of using the lower layers of the SS7 stack.

For experimentation purposes, we only execute one instance of the connection server for this group of ATM switches (all assumed to be in one peer group), though, in general, multiple instances of the connection server can be executed in the same peer group.

4 Measurements

Having implemented a proof-of-concept prototype of the PCC algorithm, the next phase is to characterize the performance of this approach. We created scripts to set up and release bidirectional two-party connections, and measured processing times for different functions (subroutines) in the PCC software. Processing time measurements are taken using the Quantify software tool from Pure Software Inc. The connection server and switch resource servers are executed on SPARCstation 10 hosts running SUNOS 4.1.3. A processing model for an intra-peer group (involving only one connection server) PCC connection setup procedure is shown in Fig. 3. The model for the release procedure is shown in Fig. 4. We use the notation $CS_i^j$ to represent the $i^{th}$ processing session in the connection server and $SRS_k^j$ to represent the $j^{th}$ processing ses-

1. MINT (Multimedia Integrated Network for Telecommunications) was an ATM research project in AT&T Bell Labs.

![Fig. 3 Model for PCC connection setup](image)

![Fig. 4 Model for PCC connection release](image)

<table>
<thead>
<tr>
<th>Time</th>
<th>Processing Time (in μsec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CS_1^1$</td>
<td>$107.4 + t_R + 112.4 + t_{rcv} + t_{snd}$</td>
</tr>
<tr>
<td>$CS_2^2$</td>
<td>$252.2 + s + s(t_{snd} + t_{rcv})$</td>
</tr>
<tr>
<td>$CS_3^3$</td>
<td>$101.2 + s + 178.4 + s(t_{rcv} + t_{snd})$</td>
</tr>
<tr>
<td>$CS_4^4$</td>
<td>$365.36 + t_{rcv} + 2t_{snd}$</td>
</tr>
<tr>
<td>$CS_5^5$</td>
<td>$281.23 + 89.19 + s(t_{rcv} + (s + 2)t_{snd})$</td>
</tr>
</tbody>
</table>
dependently processing times are listed in Table 3. Processing times $t_{snd}$ and $t_{rcv}$, the times to send and receive a UDP/IP message respectively, are dependent on the operating system. The time for route computation $t_R$ is dependent on the size of the network, and the time to program a switch (i.e., perform Steps 3 and 4 described in Section 2), defined as $t_{prog}$, is dependent on the switch hardware. We measured these four quantities separately and listed them in Table 3.

To measure $t_{snd}$ and $t_{rcv}$ accurately, we used the UNIX™ function call getrusage to measure the time for system calls, sendmsg and recvmsg, used to send and receive UDP messages. While measuring the processing time data in the connection server and switch resource servers using the Quantify tool, we excluded system calls due to accuracy problems. The parameters $t_{snd}$ and $t_{rcv}$ are environment dependent since in other implementations, TCAP transactions may be transported over S-AAL (Signaling ATM Adaptation Layer) and ATM layers rather than on UDP/IP. This will impact the times to send and receive a message.

The route computation time $t_R$ was measured to be 33.94 μs in our prototype network consisting of seven switches. For larger networks, if Dijkstra’s algorithm is used to perform on-the-fly route computation, this time could be as large as 15ms as reported in [9] for a 200-node network. We thus carry out a sensitivity analysis based on this variation of $t_R$.

Finally, $t_{prog}$ is switch hardware-dependent. In our implementation, we measured this time for two types of switches. For the MINT switch, the switch resource server is executed as a VxWorks (a real-time operating system) task running on an IPX00 processor. This task directly sends control ATM cells to program the switch fabric. The measurement indicates that a bidirectional fabric configuration can be done in 1.16ms. On the other hand, to program the switch fabric of the VP switch shown in Fig. 2 for a bidirectional connection, we require 15ms, since it uses a message-based interface (on an RS-232 serial link) between the processor executing the switch resource server and the shelf controller that, in turn, programs the fabric.

### 5 Analysis

In order to characterize the benefits of the parallel connection control (PCC) approach over the switch-by-switch sequential setup approach (referred to as the “PNNI approach”) for use within a peer group, we construct models for the latter. Estimates of processing times for the PNNI approach are obtained from the PCC measured service times presented in Section 4. Given the service times and call flows for the two models, we can carry out a queueing analysis for the end-to-end connection setup delay for different call arrival rates. In addition to the queueing delay and service times at the control processors, link emission delays are also included. Message length details are not provided in this paper due to space considerations, but can be obtained from the authors. A signaling link rate of 1.5Mbps is assumed.

Using this combination of measured service times for PCC, estimated service times for PNNI, and estimated link emission delays, we carry out a comparative delay-throughput analysis, and also compare the theoretical maximum throughputs in the two approaches. This section is organized as follows: Section 5.1 describes the models of the PNNI signaling setup and release procedures and lists estimates of the processing times for the PNNI approach; Section 5.2 describes the analysis methodology; and Section 5.3 presents the results of the analysis.

#### 5.1 Model of the switch-by-switch sequential approach (the PNNI approach)

The PNNI signaling procedure [1], illustrated in Fig. 5, is an example of a switch-by-switch sequential approach. In Fig. 5, $SW_1$ represents a typical originating switch; $SW_2$ a typical terminating switch; and $SW_3$ a typical transit switch. If there are more than 3 switches in a connection, all the transit switches between the originating switch and the terminating switch will behave like $SW_2$. EA and EB represent the ends of the connection, i.e., end hosts A and B, respectively.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{snd}$</td>
<td>240μs</td>
</tr>
<tr>
<td>$t_{rcv}$</td>
<td>220μs</td>
</tr>
<tr>
<td>$t_R$</td>
<td>33.94μs to 15ms</td>
</tr>
<tr>
<td>$t_{prog}$</td>
<td>1.16ms to 15ms</td>
</tr>
</tbody>
</table>
Models for the setup and release of a connection using the PNNI approach are shown in Figs. 6 and 7, respectively. We use the notations $SW_j^i$ and $SW_j^r$ to represent the $i^{th}$ processing session at switch $j$ during the connection setup and release phase, respectively. The service times for different processing sessions (shown in Figs. 6 and 7) in the three types of switch resource servers are shown in tables 4 and 5 for setup and release, respectively. We did not implement PNNI signaling. Instead, we estimated the times for these processing sessions using the data obtained from the PCC implementation measurements. Effectively, since many of the actions that need to be performed are the same in both models, this estimation should be accurate.

<table>
<thead>
<tr>
<th>Table 4: PNNI connection setup processing times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>$SW_1^1$</td>
</tr>
<tr>
<td>$SW_2^1$</td>
</tr>
<tr>
<td>$SW_1^2$</td>
</tr>
<tr>
<td>$SW_2^2$</td>
</tr>
<tr>
<td>$SW_3^2$</td>
</tr>
<tr>
<td>$SW_1^3$</td>
</tr>
<tr>
<td>$SW_2^3$</td>
</tr>
<tr>
<td>$SW_3^3$</td>
</tr>
<tr>
<td>$SW_1^4$</td>
</tr>
<tr>
<td>$SW_2^4$</td>
</tr>
<tr>
<td>$SW_3^4$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5: PNNI connection release processing times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>$SW_1^1$</td>
</tr>
<tr>
<td>$SW_2^1$</td>
</tr>
<tr>
<td>$SW_3^1$</td>
</tr>
<tr>
<td>$SW_1^2$</td>
</tr>
<tr>
<td>$SW_2^2$</td>
</tr>
<tr>
<td>$SW_3^2$</td>
</tr>
</tbody>
</table>

5.2 Analysis methodology

The queuing analysis for both the PCC and PNNI approaches are done by representing the system as a queueing network. The processing elements, Connection Servers (CSs), Switch Resource Servers (SRSs) in the PCC approach, Switches (SWs) in the PNNI approach, and the transmission elements (the links interconnecting the processing elements) are represented as nodes of the queueing network. The delay analysis at each individual node is done through a decomposition assumption which essentially treats each node as an M/G/1 queue, i.e., each processing/transmission session at a node is represented as an independent Poisson arrival. The critical path of the
connection setup delay is treated as an activity graph. The end-to-end delay distribution is obtained by inverting the Laplace transform of the distribution which in turn is obtained as the product of the Laplace transforms of individual distributions assuming independence among them. See [10] for details of the procedure.

It is assumed that there are a large number of switches in the system, and, on the average, a call involves \( N \) switches. However, the particular call for which we do delay analysis involves \( N_p \) switches. In general, \( N \) and \( N_p \) need not be the same. Also, we assume that each switch of the system is equally likely to be chosen in a particular call so that the load is evenly balanced across the switches.

All processing on the critical path of the connection setup delay is done serially in the PNNI architecture. However, as shown in Fig. 3, there is a lot of parallelism in the PCC architecture which we take into account in the models. As an example, consider the processing phase that starts at \( CS^1 \) and ends at \( CS^3 \) in Fig. 3, which represents the network portion of the connection setup delay (i.e., excluding the end host portions). In \( CS^1 \), \( s \) messages are sent to the \( SRS_s \), which perform the processing session \( SRS^1 \) in parallel. As part of this session, each \( SRS \) sends one message, which is received in \( CS^2 \). In addition to \( CS^2 \), there is a second source of parallelism in \( CS^3 \) and \( CS^3 \), since we assume that by the time the last message arrives at \( CS^3 \) or \( CS^3 \), they have already processed the earlier ones. Therefore, referring back to Table 1, the critical path of the connection setup delay includes all of the processing work in \( CS^2 \), \( t_{11} \) of the transmission piece of the processing work in \( CS^2 \) and \( CS^3 \), and only a fraction (corresponding to just \( 1 \)) of the receiving piece of the processing work in \( CS^3 \) and \( CS^3 \). Specifically, the total processing piece of work in \( CS^2 \) and \( CS^3 \) are \( s(t_{11}+t_{recv}) \) \( \mu \)s (this is not separated out in Table 1 for the sake of simplicity) and \( s(t_{11}+t_{recv}) \) \( \mu \)s, respectively. Of these, the pieces appearing on the critical path of connection setup delay are only \( t_{11}+t_{recv} \) \( \mu \)s and \( t_{11}+t_{recv} \) \( \mu \)s, respectively. In addition to processing delay, there are, of course, queuing delays which are appropriately taken into account.

In the PCC approach, it is assumed that there are as many Connection Servers (CS) in the system as there are Switch Resource Servers (SRSs), and the load is evenly balanced among the CSs and among the SRSs. Also, the processing power (MIPS, Millions of Instructions Per Second, for the same set of applications) of a CS is the same as that of a SRS.

5.3 Analysis results

We present the variation of the mean and 95th percentile of the end-to-end connection setup delay with the:
- throughput (call arrival rate),
- number of switches in the connections, and
- the environment-dependent parameters, \( r_e \), the route computation time, and \( t_{prog} \), the switch programming time.

Cases 1-4 represent four sets of values for the parameters \( r_e \), \( t_{prog} \), and \( N \) (in all cases, we assume \( N \), the average number of switches on a connection, to equal the number of switches on a particular connection, \( N_p \)):
- Case 1: \( (r_e = 33.94\mu s, t_{prog} = 1.16ms) \), and \( N = N_p = 2 \) switches,
- Case 2: \( (r_e = 33.94\mu s, t_{prog} = 1.16ms) \), and \( N = N_p = 10 \) switches,
- Case 3: \( (r_e = 15ms, t_{prog} = 15ms) \), and \( N = N_p = 10 \) switches, and
- Case 4: \( (r_e = 15ms, t_{prog} = 15ms) \), and \( N = N_p = 5 \) switches.

Fig. 8 shows the delay-throughput results for cases 2 and 3, while Table 6 shows the theoretical maximum throughput achievable in each of the four cases at 100% utilizations of the various servers (CS and SRS in the PCC approach, and SW in the PNNI approach). It is to be noted that the theoretical maximum throughputs shown in Table 6 are not achievable in practice since queuing delays approach infinity at those throughput values. However, they show which of the two approaches can sustain higher traffic load.

Finally, Fig. 9 shows the variation of the mean and 95th percentiles of the end-to-end setup delay with varying \( N_p \), the number of switches in a particular connection, while keeping throughput (call arrival rate), \( \lambda \), and the mean number of switches per connection, \( N \), fixed for the following two cases:
- Case 5: \( (r_e = 33.94\mu s, t_{prog} = 1.16ms) \), \( \lambda = 150\) calls/sec/switch, and \( (N = 5) \),
- Case 6: \( (r_e = 15ms, t_{prog} = 15ms) \), \( \lambda = 35\) calls/sec/switch, and \( (N = 5) \).

Description of Fig. 8:

Fig. 8 represents the cases when all connections are routed through 10 switches. Even at a low load of 10 calls/sec/switch, the mean setup delay for PCC (14.29ms) is about 34% of the PNNI approach (41.45ms) for 10-switch connections. At higher load, the difference increases. The end-to-end setup delay (mean and 95th percentile) for Case 3 is greater than for Case 2 for both the PNNI and PCC approaches, as seen in Fig. 8. This is because the larger values of \( r_e \) and \( t_{prog} \), shown in Table 3, are used for Case 3. As mentioned earlier, the route computation time, \( r_e \), is sensitive to the size of the network and the
Table 6: Theoretical maximum throughput at 100% server utilizations

<table>
<thead>
<tr>
<th>Case</th>
<th>Throughput at 100% server utilization (calls/sec/switch)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PCC</td>
</tr>
<tr>
<td>Connection Server</td>
<td></td>
</tr>
<tr>
<td>Case 1</td>
<td>319.68</td>
</tr>
<tr>
<td>Case 2</td>
<td>500.60</td>
</tr>
<tr>
<td>Case 3</td>
<td>286.19</td>
</tr>
<tr>
<td>Case 4</td>
<td>189.63</td>
</tr>
</tbody>
</table>

The switch resource management time, $t_{prog}$, is sensitive to the type of ATM switch hardware and device driver used. Compare Cases 2 and 3, both of which use 10-switch connections, at a loading of 10 calls/sec/switch. As stated earlier for Case 2, the PCC setup delay is 34% of the PNNI setup delay at this load. In Case 3, at this load, the PCC mean end-to-end setup delay is 46.72ms, which is only 20% of the PNNI mean end-to-end setup delay (230.08ms). Thus, the reductions in setup delay using the PCC approach is more significant the greater the switch programming time.

**Description of Table 6:**

We now consider the issue of throughput. Table 6 shows the theoretical maximum throughput that can be achieved in Cases 1 to 4. We discuss four observations noted from the results of Table 6:

- The difference in total throughput between the two approaches;
- The difference in per-switch throughput between the two approaches;
- The variation of the theoretical maximum throughputs with the number of switches on connections; and
- The effect of the environment-dependent variable $t_{prog}$ (switch programming time) on the theoretical maximum throughputs.

First consider the difference in the total throughput of the two approaches (PCC and PNNI). As mentioned in Section 5.2, the PCC approach has as many CSs as there are SRSs, and the processing power of a CS is the same as that of an SRS. This implies that the total processing capacity (measured, for example, in MIPS) assumed for the PCC approach is two times compared to the total processing capacity assumed for the PNNI approach. The corresponding increase in throughput should be two-fold if exactly the same amount of work is done in both approaches. However, this is not true. Both approaches accomplish the same overall task of setting up and releasing connections. However, for connection setup of $n$-switch connections, the total work in the PNNI approach consists of performing route computation once, performing switch resource management functions (consisting of the four steps listed in Section 2) $n$ times, and handling $4n$ incoming and $4n$ outgoing signaling messages. In the PCC approach, besides all this work (distributed between the connection server and switch resource servers), the connection server receives an additional 2 messages and sends an additional 3 messages. However, for connection release, the PCC procedure is simpler than the PNNI procedure (see Figs. 4 and 7). In PCC, we simply use unidirectional messages for release assuming that the lower layer of the protocol stack ensures reliable delivery of messages.

These two effects, the larger setup processing and the smaller release processing, work at cross-purposes, causing the total throughput numbers to be dependent on the number of switches traversed by connections. In the PCC approach, given that the CSs and SRSs reach saturation at different loads, the relative numbers of these servers have to be varied to make them both equally loaded. For example, in Case 2 (see results in Table 6), by reducing the number of connection servers by a factor of $(342.99/500.60)$, both connection servers and switch resource servers will achieve the same theoretical maximum throughput at 342.99 calls/sec. This implies that one unit of processing power in the PCC approach will yield a theoretical maximum throughput of $342.99 / (1 + (342.99/500.60))$, which is equal to 203.5 calls/sec. On the other hand, one unit of processing power

---

2. The terminating switch receives 1 less incoming message, as seen in Fig. 5. Presumably, this effect is small.
in the PNNI approach yields a theoretical maximum throughput of 190.63 calls/sec. Thus, the PCC approach yields a higher total throughput for Case 2. This is true for Case 3 (49.38 calls/sec for PCC vs. 48.59 calls/sec for PNNI), and Case 4 (45.39 calls/sec for PCC vs. 45.31 calls/sec for PNNI) also.

However, in Case 1 (two-switch connections), using a similar computation, one unit of processing power in the PCC approach yields the theoretical maximum throughput of 165.46 calls/sec/switch, while in the PNNI approach, the corresponding figure is 192.05 calls/sec/switch. For connections with a small number of switches, the reduced release processing in PCC is dominated by the processing of the extra five signaling messages needed for connection setup. If we remove the effect of the reduced processing for connection release (under the assumption that the PNNI release procedure can be changed to only use unidirectional messages), then we note that the processing per connection setup in PCC is more than that for PNNI by a fixed amount, which is independent of the number of switches in the connection. Thus, in the limit, as the number of switches per connection approaches infinity, the total theoretical maximum throughput of both approaches will be the same.

Next, consider the second throughput measure: the per-switch theoretical maximum throughput. As seen in Fig. 8, the PCC approach achieves higher throughput, i.e., the processors can sustain a higher load (calls/sec/switch) before reaching saturation than in the PNNI approach (also seen in Table 6). The reason for this behavior is that, in the PNNI approach, a switch performs many more functions than a switch resource server in the PCC approach. Thus, the theoretical maximum throughput is higher for an SRS in the PCC approach than for an SW in the PNNI approach.

In the PCC approach, there are two types of servers, CSs and SRSs. The connection servers also do not reach saturation as fast as the PNNI switch processors. In the case of connection servers, each server does perform more work per connection than a switch in the PNNI approach. Based on the data shown in Tables 1 and 4, the total work done at a connection server per connection setup is greater than at a PNNI switch. However, the call arrival rate experienced by a connection server is less than that experienced by a PNNI switch. Since each connection requires $s$ switches, and just one CS, and we assume the same number of CSs and SRSs in the system (with the load evenly distributed among all CSs and SRSs), the call arrival rate witnessed by a CS is only $1/s$ of the per-switch call arrival rate. Thus, in the PCC approach, even though a connection server has more work per connection setup than a PNNI switch, it still reaches 100% utilization at much higher per-switch call arrival rate than a switch in the PNNI approach, as shown in Table 6.

The third observation from the results presented in Table 6 is about the variation of the theoretical maximum throughput with the number of switches in connections. As $s = N = N_p$ increases from 2 to 10, the theoretical maximum throughput of the SRS remains unchanged while that of the CS goes up as seen in Table 6. As a result, the system bottleneck in the PCC approach shifts from the CS to the SRS as we move from Case 1 ($S = 2$) to Case 2 ($S = 10$). We explain the reason behind this phenomenon as follows.

Let $\lambda_{CS}$, $T_{CS}$ and $\rho_{CS}$ represent the call arrival rate, the processing time per call and the load per call, respectively at the CS. Let $\lambda_{SRS}$, $T_{SRS}$ and $\rho_{SRS}$, represent the corresponding quantities at the SRS. Clearly,

$$\rho_{CS} = \lambda_{CS} T_{CS} \quad \text{and} \quad \rho_{SRS} = \lambda_{SRS} T_{SRS}. \quad \text{EQ 1}$$

By definition, for a fixed calls/sec/switch, the quantity $\lambda_{SRS}$ is independent of $s$. From Table 2, we see that $T_{SRS}$ is also independent of $s$. Therefore, the load $\rho_{SRS}$ is independent of $s$ as well. Now, since each call requires $s$ SRSs, and just one CS, and we assume the same number of CSs and SRSs in the system (with the load evenly distributed among all CSs and SRSs), we see

$$\lambda_{CS} \propto \frac{1}{s} \quad \text{EQ 2}$$

From Table 1, we see that,

$$T_{CS} = a + bs \quad \text{EQ 3}$$

where $a$ and $b$ are positive constants. Combining (EQ 2) and (EQ 3), we get,

$$\rho_{CS} = \lambda_{CS} T_{CS} \propto b + \frac{a}{s}. \quad \text{EQ 4}$$

Therefore, the load at a CS decreases with increasing $s$ for a fixed calls/sec/switch. This explains why at lower values of $s$, the CS is more loaded than the SRSs, but at higher values of $s$, CS becomes less loaded than the SRSs, causing the bottleneck to shift with increasing $s$.

As for the switch in the PNNI approach, the theoretical maximum throughput may increase or decrease with the number of switches on connections, based on the values of $t_R$ and $t_{prog}$. For the first two cases, there is a slight reduction in the number of calls that can be handled before the switch processor is 100% utilized in Case 2 relative to Case 1. The reason for this difference is that the end switch processes one less segment (see Fig. 6: $SW_3$ has only three processing segments, while $SW_1$ and $SW_2$ have four each) compared to other switches. If the number of switches per connection increases, then, on average, fewer switches would act as terminating switches, and hence the average processing load per switch would go
down slightly. On the other hand, for the last two cases, there is a slight increase in the number of calls that can be handled before the switch processor is 100% utilized in Case 3 relative to Case 4. The reason for this behavior is that route computation time \( t_R \) is only needed in the originating switch. This is significant (15 ms) compared to the reduced terminating switch processing time (321.7 \( \mu \)s). Thus, in Case 3, since there are more switches, fewer switches act as originating switches, and hence the average processing load per switch would increase. In Case 1 and Case 2, \( t_R \) is only 33.94 \( \mu \)s, and hence the dominant effect is that of reduced load of terminating switches.

The final observation made from the results presented in Table 6 is the effect of the environment-dependent variable \( t_{prog} \) (switch programming time) on the theoretical maximum throughput. The servers reach 100% utilization at much lower call arrival rates for Cases 3 and 4, relative to Cases 1 and 2, due to the increased service times required for route computation and switch programming tasks. For higher values of the switch programming time \( t_{prog} \), the switch resource server becomes the bottleneck. Hence, the difference between the per-switch theoretical maximum throughput that can be achieved using PCC and PNNI becomes smaller at higher values of \( t_{prog} \). However, as already explained, the gain in setup delay increases at these higher values of \( t_{prog} \). Thus, at lower values of \( t_{prog} \), a much higher per-switch theoretical maximum throughput can be achieved with PCC, while the end-to-end setup delay reduction for PCC (relative to PNNI) is not as high, while the opposite is true at higher values of \( t_{prog} \).

Description of Fig. 9:

Fig. 9 shows the variation of the mean and 95th percentiles of the end-to-end connection setup delay with the number of switches on the route of a connection, for different values of \( t_R \) and \( t_{prog} \) (Cases 5 and 6, as defined at the start of this section). In Cases 1-4, we assumed that all connections are routed through the same number of switches. Thus, in Case 1, all connections are routed through 2 switches, in Case 2, through ten switches, etc. However, in a typical situation, different connections will be routed through a varying number of switches. To handle this, as stated earlier (Section 5.2), we create two variables \( N \) and \( N_p \), to represent the average number of switches in a connection, and the number of switches in a particular connection, respectively. From a throughput perspective, if a switch experiences an arrival rate of \( \lambda \) calls/sec, each connection server will experience \( \lambda/N \) calls/sec (EQ 2, where \( s = N \)). For Cases 5 and 6, we keep \( \lambda \) and \( N \) fixed, and therefore, the load at all servers is fixed as well, but just study how the end-to-end connection setup delay varies with \( N_p \), the number of switches in a particular connection.

Case 5 of Fig. 9 assumes the smaller values of \( t_R \) and \( t_{prog} \), and sets the switch throughput at 150 calls/sec. The mean number of switches per connection is assumed to be 5. Thus, a connection server experiences 150/5 or 30 calls/sec. At this load, the connection server utilization is 34.2% and switch resource server utilization is 43.7% in the PCC solution. The switch in the PNNI solution is 78.5% loaded at this throughput.

Case 6 of Fig. 9 shows the variation of setup delay with \( N_p \) for the higher values of \( t_R \) and \( t_{prog} \). Since we assume \( N = 5 \), the results for Case 4 can be consulted to select an appropriate call arrival rate. At a loading of 35 calls/sec/switch, servers in neither approach are as yet saturated. Hence, we use this call arrival rate to study the variation of the end-to-end setup delay with \( N_p \) for Case 6. At this load, the connection server utilization is 18.5% and switch resource server utilization is 58.6% in the PCC solution. The switch in the PNNI solution is 77.2% loaded at this throughput.

The major observation from Fig. 9 is that in the PNNI approach the end-to-end setup delay increases fast with \( N_p \), whereas the rate of increase is much slower in the PCC approach. The reason for this difference lies in the increased service time caused by sequential processing rather than parallel processing. With both approaches, the load is fixed since calls/sec/switch and \( N \) are fixed. This implies that the queueing delay during each visit at a server does not change. Given the sequential nature of the PNNI approach, and the fact that \( N_p \) switches are visited, each time with fixed processing and queuing delays, the mean connection setup delay is roughly of the form \( aN_p \), where \( a \) is a positive constant. By contrast, in the PCC approach, due to the inherent parallelism of work among
the SRSs (Section 5.2), and also due to further parallelism of work at the CSSs, the mean connection setup delay is roughly of the form $aN_p + b$, where $a$ and $b$ are positive constants with $a < b$.

Case 5 of Fig. 9 shows that for a given 95th percentile end-to-end setup delay requirement of 50 ms, a connection can be routed through a maximum of 3 switches in the PNNI approach, while in the PCC approach, the limit is several hundreds. In Case 6 of Fig. 9, for a given 95th percentile end-to-end setup delay of 300 ms, no more than 2 switches must be located on the route of the connection in the PNNI approach, while there is practically no upper limit in the PCC approach even if the delay requirement is only two-thirds as much (i.e., 200 ms). The implication of these results is that a reduced number of provisioned virtual path connections (used to decrease the number of switches that need setup for on-demand connections) is required in the PCC approach relative to the PNNI approach in order to meet the specified end-to-end delay requirement. The use of fewer provisioned VPCs leads to better network utilization for the PCC approach relative to the PNNI approach as was demonstrated in [2].

6 Conclusions

This paper described an implementation of PCC (Parallel Connection Control), reported measured service times (software execution times) obtained from this implementation, and used the measured data in a queueing analysis to quantify the benefits of PCC over PNNI (the standards approach) signaling. At low loads (removing the effect of queueing delays), the PCC end-to-end setup delay is only 20% of the PNNI setup delay for the larger values of the route computation and switch programming times, and it is only 34% of the PNNI setup delay for the smaller values of these times, assuming ten-switch connections. The difference between the two types of delays is much more at higher loads. Thus, significant gains in end-to-end setup delay are obtained by exploiting two forms of parallelism (across switch resource servers on a connection, and between connection servers and switch resource servers). The PCC end-to-end setup delay grows much slower with the number of switches on connections than the PNNI setup delay. This allows far fewer VPCs (Virtual Path Connections) to be provisioned in the PCC approach, leading to better network utilization. To achieve a 300-ms 95th percentile end-to-end setup delay, in the PNNI approach, connections can only pass through 2 switches (assuming the higher values of the route computation and switch programming times). On the other hand, there is no discernible limit for PCC.

The per-switch throughput of the PCC approach is 80% more than that of the PNNI approach for smaller values of the route computation and switch programming times, and 23% more than that of the PNNI approach for larger values of these times, assuming ten-switch connections (assuming equal processing capacities for the PNNI switch and PCC switch). As for the total throughput, PCC and PNNI are about the same in all cases, given that the same overall "work" of setting up and releasing connections is performed.

7 Acknowledgment

We thank Clark Woodworth, W. H. (Tony) Lau, and Kai Eng for providing us the hardware ATM platform on which we implemented the PCC software. We also thank Sunil Malhotra, Sizheng Wei, Tom La Porta, Ram Ramjee, and Maribel Lopez, for helping with the software implementation.

References