Purdue University
Purdue e-Pubs

Computer Science Technical Reports | Department of Computer Science

2001

Intelligent Grouping of TCP Flows for Coordinated Congestion Management

Hoi Chang

Rajeev Gopalakrishna

Venkatesh Prabhakar

Report Number:
01-017

Chang, Hoi; Gopalakrishna, Rajeev; and Prabhakar, Venkatesh, "Intelligent Grouping of TCP Flows for Coordinated Congestion Management" (2001). *Computer Science Technical Reports.* Paper 1514.
http://docs.lib.purdue.edu/cstech/1514

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

INTELLIGENT GROUPING OF TCP FLOWS FOR COORDINATED CONGESTION MANAGEMENT

Hoi Chang
Rajeev Gopalakrishna
Venkatesh Prabhakar

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD TR #01-017
September 2001

Intelligent Grouping of TCP Flows for Coordinated Congestion Management

Hoi Chang, Rajeev Gopalakrishna, Venkatesh Prabhakar
Purdue University, West Lafayette, IN 47906

December 11, 2000

Abstract

End-to-end congestion control mechanisms help utilize network resources more optimally because they inherently cooperate with the congestion control mechanisms inside the network. Recent research has shown that coordinating congestion control strategies between flows can improve the performance of the group of flows as a whole. An important aspect of such congestion management is identifying flows to be grouped together and the coordinated congestion technique to be adopted for the group. Currently flows between the same end systems are grouped togther. Here, we attempt to provide an efficient, dynamic grouping mechanism for a set of flows originating from an end system on the basis of shared points of congestion. We use a correlation of the TCP round trip times (RTTs) to provide an estimate of the shared points of congestion.

1 Introduction

Current congestion control mechanisms work on a per-flow basis, regulating the data sending rate in conformance with the network conditions as assessed by the flow. Recent trends in research have focused on performing coordinated congestion control at the end systems. The motivation for this approach is to provide fairness and improve performance among the flows sharing certain characteristics. Additionally, such end to end congestion control schemes are more network friendly and cooperative with the congestion control techniques in place at the center of the network.

The two key aspects of coordinated congestion control are:

- Grouping flows based on some common criterion.
- Improvising on the current TCP congestion control mechanism to make it work better for a group of flows in terms of aggressiveness and responsiveness to the network conditions.

Typically flows between the same end systems are grouped together. This is based on the observation that routes in the Internet change less frequently and hence it is reasonable to assume that most packets of flows between the same end systems traverse the same path. Hence the information about congestion in the network that is assessed by one flow can be used to control all other flows in the same group as they share the same bottleneck.

Coordination of the congestion control schemes of flows in a group is done by maintaining a common congestion window for all flows or by adjusting the congestion window of the individual flows appropriately or by adjusting the rate at which data is sent from the end system.

In our work, we attempt to provide a grouping scheme which groups TCP flows that share common points of congestion. We use a correlation of the RTTs of the TCP flows originating from an end system to identify sets of flows that share their maximally congested links. We also provide a mechanism for adjusting the congestion windows of the flows in a group based on the state of the network as assessed by all group members.

The remainder of the report is organized as follows: Section 2 summarizes the related work in this area. Section 3 explains our dynamic grouping scheme. In section 4, we explain how we coordinate the congestion management among flows in a group. Section 5 presents some of the simulation results that were performed. We conclude in section 6 observing that dynamic grouping helps in achieving fairness and better overall performance among the flows originating from an end system.

2 Related Work

TCP-Int [5]uses one common congestion window for all concurrent TCP connections which is increased upon receipt of an acknowledgement and decreased upon detection of packet loss by any connection.

Ensemble-TCP [6]also groups flows between same end systems. In addition it caches information about the measured state of network congestion for a period of time. It uses the cached information to expedite the start up phase of the new connections.

Congestion Manager [2]presents a framework for managing network congestion from an end-to-end perspective. It seperates congestion management from the transport layer and uses a common window for all flows between the same pair of end systems. Application and transport layer feedback is used to adjust this window appropriately.

[3]emphasises the need for a coordinated approach to congestion management at an end system. It identifies topology discovery,correlating delay and/or loss at the receiver and enhanced ECN as means of detecting shared bottleneck links among flows in order to group them.

[1]uses loss correlation, delay correlation and poisson probes to identify shared bottleneck links.

3 Dynamic Grouping of TCP flows

Instead of grouping concurrent TCP flows based on the same end hosts, we group TCP flows based on the correlation of their packet round-trip times. The idea is to identify and group together flows that experience common major congestion bottlenecks (regardless of where their other end hosts locate), and then to apply coordinated congestion management to the flows that face similar congestion problems on a group basis. Deciding whether or not two out-going flows originated from the same source experience same major bottlenecks is not a new problem; Rubenstein, et al in [7]have proposed some statistical solutions to it that are based on finding the correlation between the packet losses and delays of two flows. Here, we apply their concept of correlation analysis to the context of dynamically grouping concurrent TCP flows in real-time.

To make our grouping scheme dynamically reflect current congestion states of the network, correlation analysis and grouping/regrouping of flows need to be performed periodically. As shown in Figure 1, a new timer is introduced into TCP to periodically invoke the correlation analysis and regrouping of TCP flows (discarding previous grouping information). During each correlation period, TCP flows are coordinated on a group basis in their responses to different network states for that time period. See Section 4 for details on coordinating TCP flows in a group.

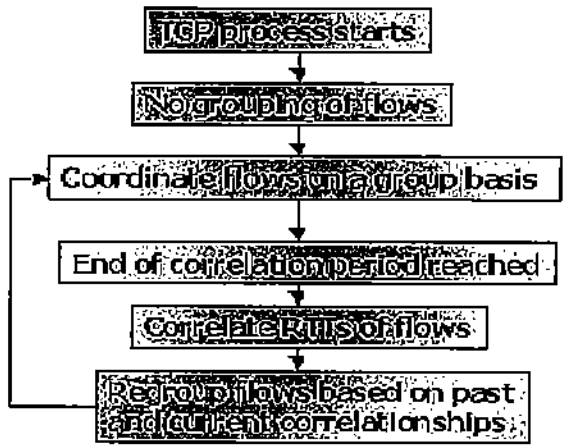

Figure 1: The process of correlating, grouping, and coordinating TCP flows

3.1 Correlating RTTs of TCP flows

We use Pearson's correlation function [8][7]for correlating both the RTTs of acked packets and the timed out durations for lost packets between two TCP flows. In his formula,

$$r_{xy} = \frac{\Sigma(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\Sigma(x_i - \overline{x})^2 \Sigma(y_i - \overline{y})^2}}$$

r_{xy} is the correlation coefficient (with range $[-1, 1]$) of the two data sets x_i and y_i whose averages are $\overline{x}$ and $\overline{y}$, respectively. The closer r_{xy} approaches $+1$ (-1), the more positively (negatively) linear the data points (x_i, y_i) are. If $r_{xy} = 0$, the data points show no linear relationship – that is, they do not correlate.

Since a TCP flow may receive its acks or experience retransmission timeouts at very different times from other concurrent TCP flows, simply selecting a random set of data values (i.e., RTT or timeout values) of the same size from each flow and then correlating them with those of other flows would very possibly yield poor or incorrect correlation results. Rather, the data values to be correlated have to be selected carefully. In Figure

2, we present a simple heuristic for pairing up data values from two TCP flows that are to be submitted to the correlation analysis: pair up the data values from two flows that are the closest (in time) to each other.

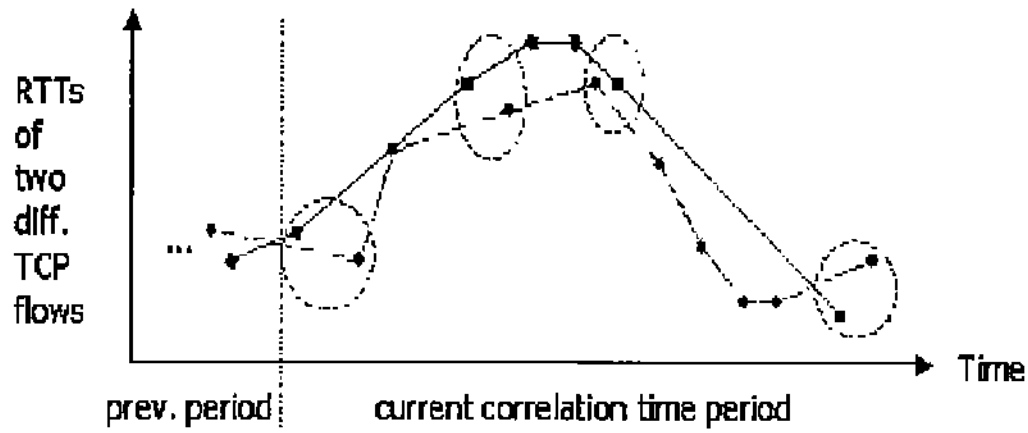

Figure 2: A heuristic for pairing up and correlating data values from two flows

3.2 Grouping decision

The grouping decision for two TCP flows is based on the value of their smoothed correlation coefficient, sr_{xy}, which is a weighted average of r_{xy} and R_{xy}, the correlation coefficient for the current correlation period and the correlation coefficient for the entire history of the two flows (including the current correlation period). In other words,

$$sr_{xy} = w * r_{xy} + (1 - w) * R_{xy}$$

If sr_{xy} is greater than a preset *threshold* value (0.75 for example), then the two flows are grouped together. In order for groups to be expandable to a larger size, two groups are further merged into a single group if the smoothed correlation coefficient between any two flows from both groups is greater than *threshold*. While this may seem to be incorrectly merging groups of flows some of which may not correlate among themselves, our grouping experiments convince us that, if *threshold* is set to a relatively high value (say 0.75), it is quite likely that all flow pairs exhibit a similarly high correlation.

The cost of deriving R_{xy} is not as expensive as it may seem. Indeed it is just $O(n)$, where n is the number of data points in a correlation period. The reason is that the computation of R_{xy} can be carried over and continued from one correlation period to the next.

3.3 Tuning the parameters

Grouping decisions depend on three control parameters:

- The length of a correlation period. If it is too long, flows are unable to be regrouped in time to reflect changes in congestion states. If it is too short, there would not be enough data points available for an accurate correlation analysis. For example, if only two data points are collected for two flows, then the points show either a perfectly positive or negative linear relationship ($r_{xy} = 1$ or -1) or no correlation at all ($r_{xy} = 0$).

  In our simulation experiments, we fixed the length of correlation periods to be several seconds long. Depending on the congestion levels experienced by the flows, there were as many as 60 data points and as less as 10 data points collected by each flow in a period of several seconds. (From our experiments, we found that 10 to 15 data points per correlation period is appropriate for an accurate, in-time correlation analysis.) A possible improvement over our fixed correlation period scheme, therefore, is to vary the length of a correlation period according to the numbers of data points already collected by the majority of flows (for example, if more than three-fourths of the flows have collected 15 data points each, then the correlation period ends even when the timer has not timed out yet).

- The weight w for computing r_{xy}. If w is too large, r_{xy} dominates the grouping of flows. This may not be ideal as correlation coefficients computed over short time durations are less precise and thus grouping of flows based almost entirely on these values may be highly unstable. On the other hand, if R_{xy} carries a larger weight than r_{xy}, the flow grouping mechanism would not be able to promptly react to new changes in network congestion. We fixed $w = 0.75$ in our simulation experiments.

- The *threshold* for grouping flows. From our experience, *threshold* $= 0.8$ or above allows flows that really correlate (i.e., that share the same major bottlenecks) to be grouped together. However, when flows are newly started, their correlation coefficients for the beginning correlation periods are usually unstable due to a relatively short correlation history. A possible improvement over the fixed-threshold scheme is to set *threshold* higher momentarily for (only) newly started flows.

4 Coordinating TCP flows in a Group

The two operations performed on the congestion window of a TCP flow are increasing it in the event of receiving acknowledgements and decreasing it upon detecting packet loss. The amount by which the congestion window is varied depends on the phase the TCP session is in, namely slow start or congestion avoidance or fast retransmit (in later versions like Reno, New Reno etc.). In our approach, TCP flows within a group are coordinated by adjusting the congestion window of all flows based on the assessed network state by a single flow.

We modify the window updation functions to perform:

- Upon packet loss for a flow, decrease the congestion window (*cwnd*) and *ssthresh* for all other flows in the group accordingly.

- Upon receipt of a certain number of acks by the flows in a group, whose RTTs are lower than the smoothed RTTs of the flows, increase the congestion window (*cwnd*) of all other flows in its group.

We reduce the congestion window of all flows in a group whenever a flow detects a packet loss because we want to make all flows react instantly to the congestion in the network.

Here the *cwnd* and the *ssthresh* of the other flows are modified in the same manner as they are modified for the flow that detected congestion.However this can be improved by basing the modifications on the relative state of the flows compared to the flow that detected congestion. On the other hand, we use a count of the number of flows that have received their acknowledgements for the data that was sent and only when the count is beyond a threshold, do we increase the congestion window of all other flows. This is because we want to react less aggresively to flows detecting available bandwidth. Otherwise, all TCP flows would become overly aggressive in transmitting data leading to more congestion and in turn more packet losses. The threshold can be set to half the number of flows which would result in increasing the congestion window of flows based on the majority of the flows in the group obtaining acknowledgements. Also the threshold can be set to a fixed number. We have found 2 as a suitable threshold in our experiments with 6 TCP flows.

Our grouping scheme is independent of the way in which we coordinate the congestion windows and it can be used with any other coordinated congestion control scheme which is based on grouping flows.

5 Simulation Results

We have performed simulation experiments on the ns-2 simulator. The topology used is shown below.

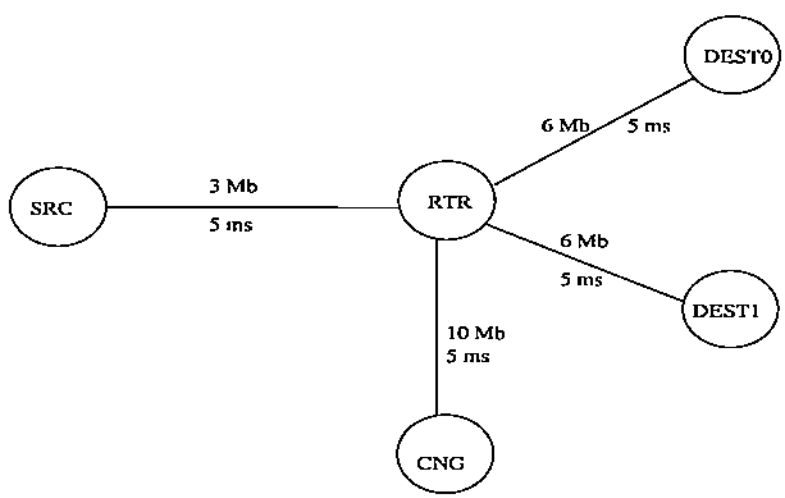

Figure 3: Topology for Simulation Setup

src is the source of traffic (FTP and cross traffic), *dest*0 and *dest*1 are the sinks for the FTP traffic, and *cng* is the sink for the cross traffic. Six FTP flows were used, three each between *src* and *dest*0 and *dest*1 over TCP New Reno connections. The FTP agent attached to *src* transmits an infinitely long file, thus it uses the TCP connection for the entire length of experiment. Pareto cross traffic was introduced between *src* and *cng*. We used Pareto cross traffic because recent research suggests that it models the Internet traffic closely. The length of the experiment was 100 seconds. The peak rate of the Pareto cross

traffic was set to $500Kbps$, which serves as a good measure of the congestion in the link shared by all six flows

We performed three simulation experiments whose results are described below.

In the first case, the link between *src* and *rtr* is the maximum bottleneck link for all the six flows. Here we found that all the six flows were grouped together most of the time. The plots of the sequence number of the acknowledgements received at *src* versus time is shown in Figures 4 and 5, for both our method and that by using normal TCP New Reno without using coordinated congestion control. (In all cases, the x-axis is the time in seconds and the y-axis is the sequence number).

We observe that the throughput achieved by the flows in the normal case varies a lot, whereas using our scheme, they are much more uniform. The average throughput achieved by all the flows is almost the same as (but a bit lower than) that in the normal case, the difference being negligibly small. Also, packet loss rate is much lesser, which indicates a more network friendly behavior on the part of the TCP flows. This proves the correctness of our grouping scheme that flows sharing their maximally congested points must be grouped together.

In the second case, Pareto traffic with a peak rate of 6 Mbps is additionally introduced between *rtr* and *dest*0 making it more congested than the link between *src* and *rtr*.

We observe that flows between *src* and *dest*0 are grouped together and those between *src* and *dest*1 are grouped together. This grouping is correct because the maximum point of congestion for the flows between *src* and *dest*0 occurs on the link between *rtr* and *dest*0 and that it is far higher than the congestion experienced by the flows at the earlier link. Again, we see that the throughput of the flows grouped together do not vary as they do in the normal case. Figures 6 and 7 illustrate the results.

In the third case, we had the same set up as the second case, except that the Pareto cross traffic between *src* and *dest*0 is introduced after 50 seconds had elapsed in the experiment.

We observe that initially all flows are grouped together, and after 50 seconds flows to *dest*0 are grouped together, while those to *dest*1 are grouped together. This verifies that our algorithm groups flows together dynamically. Again the throughput of all the flows in a group do not vary. (See Figures 8 and 9.)

From the above simulations we infer that:

- Our dynamic grouping algorithm works correctly, i.e., flows sharing their same maximally congested points are grouped together.
- Coordinating the congestion management of the flows in a group leads to better fairness.

6 Conclusion

We have provided a mechanism to dynamically group TCP flows based on correlating their RTTs. Our correlation alogorithm identifies flows that share common points of congestion and the degree of correlation reflects the degree to which they share common bottlenecks. Our grouping mechanism thus groups flows that share their same maximally congested points. Performing coordinating congestion control among flows in a group leads to better fairness among them.

7 Future work

Grouping of UDP flows have not been considered here because of lack of RTT measurements on the part of UDP. Special probe mechanisms may need be considered in order to group them. We can also vary the congestion control algorithm and try to tune the performance of the flows.

References

[1] Dan Rubenstein, Jim Kurose, and Don Towsley, Detecting Shared Congestion of Flows Via End-to-end Measurement, Proceedings of ACM SIGMETRICS 2000.

[2] Hari Balakrishnan, Hariharan S. Rahul, and Srinivan Seshan, An Integrated Congestion Management Architecture for Internet Hosts, Proceedings of ACM SIGCOMM, August 1999.

[3] Venkata N. Padmanabhan, Coordinated Congestion Management and Bandwidth Sharing for Heterogeneous Data Streams, Proceedings of NOSSDAV 1999.

[4] Sally Floyd and Kevin Fall, Promoting the Use of End-to-End Congestion Control in the Internet, Proceedings of IEEE/ACM transactions on Networking 1999.

[5] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, M. Stemm, and R.H. Katz, TCP Behaviour of a Busy Web Server: Analysis and Improvements, Proceedings of IEEE INFOCOM 1998.

[6] L. Eggret, J. Heidemann and J. Touch, Effects of Ensemble-TCP, Computer Communications Review, January 2000.

[7] D. Rubenstein, J. Kurose, and D. Towsley, Detecting Shared Congestion of Flows Via End-to-end Measurement, Proceedings of ACM SIGMETRICS 2000, San Jose, CA, June 2000.

[8] H. M. Wadsworth, Jr., Editor, Handbook of Statistical Methods for Engineers and Scientists, 2nd Ed., McGraw-Hill, NY, 1998.

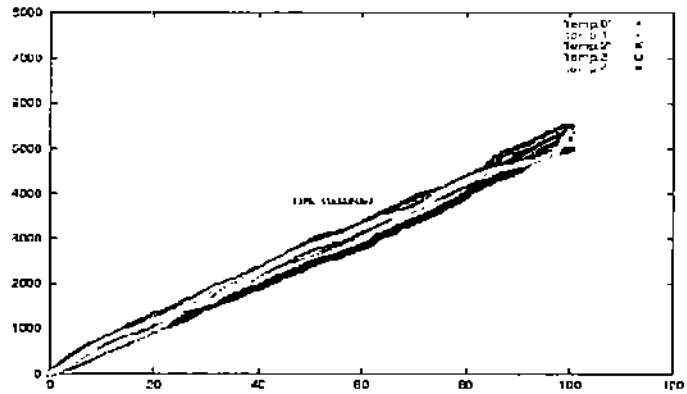

Figure 4: One bottleneck; grouping enabled. X-axis denotes time in seconds; Y-axis show the packets that have been acked.

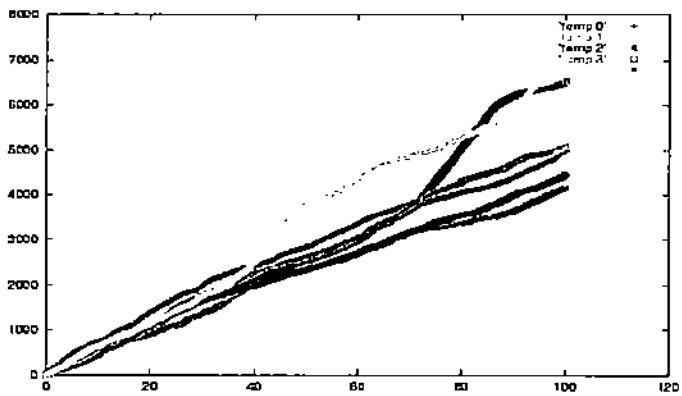

Figure 5: One bottleneck; grouping disabled.

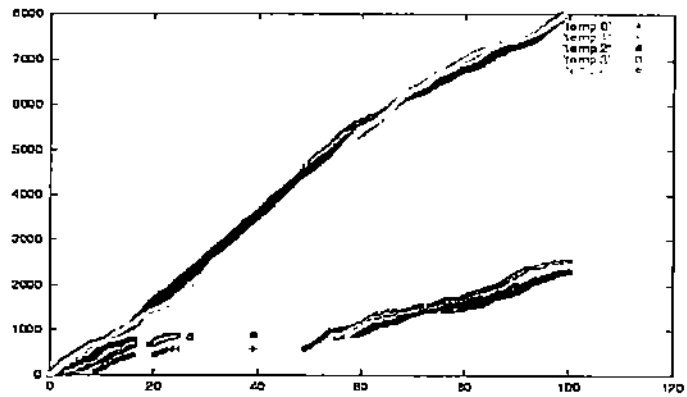

Figure 6: Two bottlenecks; grouping enabled.

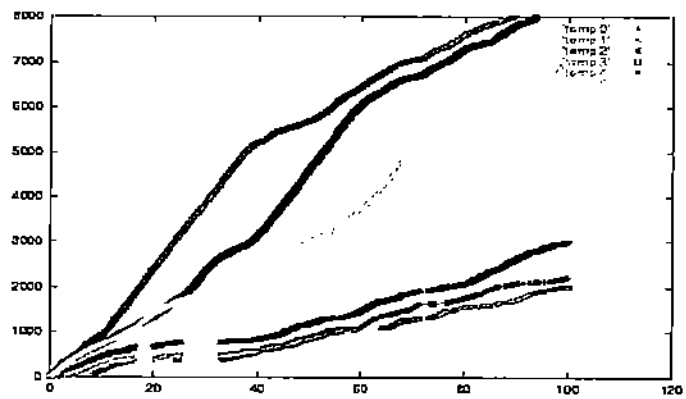

Figure 7: Two bottlenecks; grouping disabled.

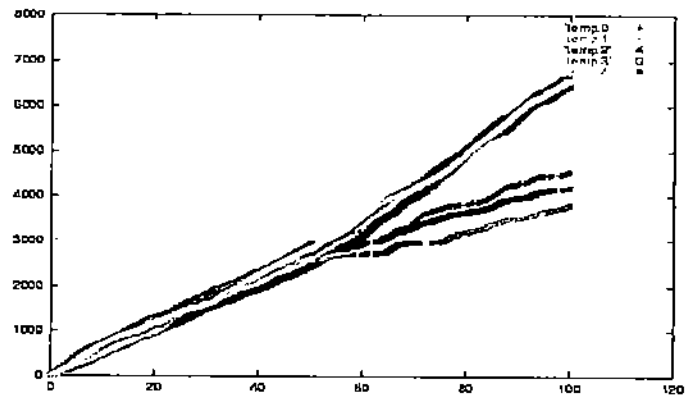

Figure 8: Second bottleneck starts half-way; grouping enabled.

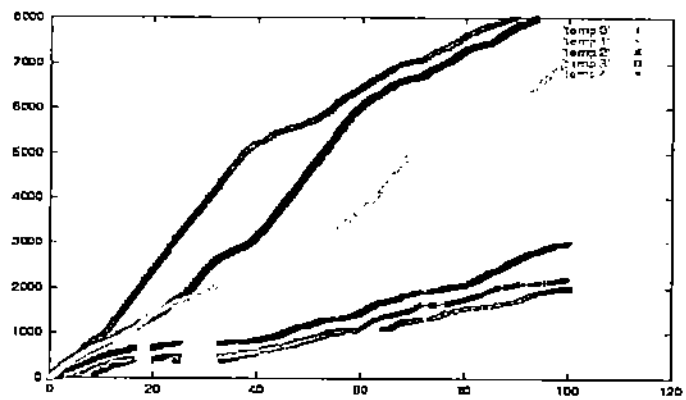

Figure 9: Second bottleneck starts half-way; grouping disabled.