Congestion Control for High Performance Virtual Cut-Through Networks *

L. Fernández
Dep. de Ingeniería y Tecnología de Computadores
University of Murcia
Facultad de Informática
Campus de Espinardo, Murcia, Spain
email:lfmaimo@ditec.um.es

José M. García
Dep. de Ingeniería y Tecnología de Computadores
University of Murcia
Facultad de Informática
Campus de Espinardo, Murcia, Spain
email:jmgarcia@ditec.um.es

ABSTRACT

Network performance in multiprocessors degrades dramatically when the network is beyond saturation. Virtual channels and adaptive routing soften this effect, but they cannot eliminate it. Moreover, close to saturation, packets become blocked faster than deadlock avoidance or recovering mechanisms can manage. In the latter case, the deadlock frequency becomes so high that it makes recovery solutions inacceptable. Network congestion effects include performance breakdown, exponential augment of packet latency and asymmetries in the use of buffers in symmetric networks, making the network inappropriate for providing QoS. Congestion control has been widely discussed in literature in wormhole contexts. However, due to the ease of integrating large buffers in switches, cut-through is becoming popular, opening a new branch of study.

To be able to deal with network congestion some global network information would be necessary. However, for propagating such information, additional resources must be used. In this paper, a congestion control technique for k-ary n-cubes with virtual cut-through is proposed. This technique uses local information for congestion detection, provided by the number of occupied buffers, to estimate the congestion level. When congestion is detected, it acts by limiting the injection of new messages in the source of the problem. This source is deduced by looking into the headers of the buffered and recently routed messages.

KEY WORDS

Network congestion control, high-performance networks, virtual cut-through

1 Introduction

Nowadays, processors are increasing their clock rate in such a way that soon the interconnection network will become the system's bottleneck. Moreover, we can connect more than one host/processor per router to make the best use of the network. In this way, the injected load increases and, as a consequence, the network can reach saturation, greatly degrading the system's performance. A mechanism that avoids reaching the saturation point would help to improve the system's throughput. Figure 1 shows how delivered load dramatically falls when the network enters in the saturation zone.

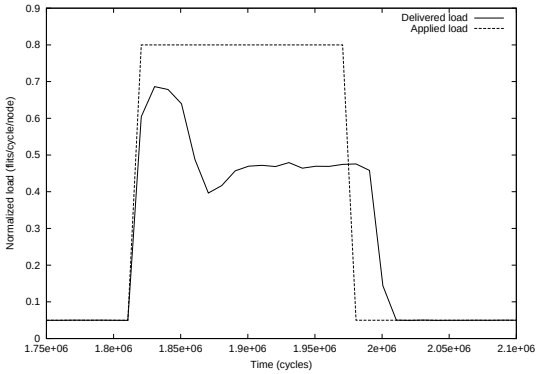

Figure 1. Effect of the congestion in the network.

Deadlocks have been well studied in literature [1, 2], leading to the conclusion that they have great influence in the performance of the network when it is saturated. On the other hand, nowadays virtual cut-through has became popular in multicomputer environments owing to the ease of integrating large buffers in switches. It offers as an advantage versus wormhole that a blocked message can be buffered in the router, releasing the resources occupied by its tail. Virtual channels can help us to postpone the apparition of congestion, by allowing other messages to advance despite some of them remaining stopped. An adaptive routing algorithm can also help, by trying to avoid the congested zones. But this just shifts the saturation point, thus, delaying the moment the network gets saturated, but not avoiding it, leading to an unbalanced virtual channel utilization. Furthermore, the point this occurs will depend on the message destination distribution

*This work has been supported in part by the spanish Ministry of Ciencia y Tecnología under project TIC2000-1151-C07-03

(for example, with bit-reversal or shuffle distributions it will be reached sooner than using a uniform one).

1.1 Congestion avoidance vs. congestion control

There exists a point in the applied load vs. time graph at which the network performance begins to decrease. This point is close to the knee of the latency curve that starts to show a slight increase. When the applied load is increased even more, the network gets saturated, and the delivered load falls sharply. This point coincides with an exponential augment of latency.

The congestion problem may be handled by avoiding the congestion or controlling it. Congestion avoidance tries not to exceed the performance knee associated with the sharp increase of the latency. On the other hand, congestion control avoids the saturation of the network by keeping the delivered load at its maximum level but suffering greater latencies.

Several methods have been proposed in literature for congestion controlling in high performance interconnection networks, but all of them are based on wormhole. The characteristics of this switching technique make it difficult to find heuristics which provide a local estimation of the congestion [3, 4, 5]. However, virtual cut-through offers new possibilities for detecting the apparition of congestion.

In this paper we present two new congestion control techniques to prevent a virtual cut-through network from reaching saturation using local information for detection purposes. These techniques work well with several destination distributions.

2 Background and related work

In wormhole contexts, several congestion control mechanisms can be found in literature. Their approaches are diverse, but we can mainly classify them in two types: those based on local information and those based on global information

The former ones use the congestion side effects in the nodes for congestion detection purposes. Usually, these methods use message throttling in the node that detected the congestion as a way of handling it, because they try to act locally as well. This has as a disadvantage that we are not acting on the responsible node, furthermore, the detecting node is penalized without cause. Lopez et al. [4, 5] measure the amount of output virtual channels occupied in the node to estimate the congestion. However, it is very difficult, if not impossible, to adjust its parameters for working well in all cases. Baydal et al. [3] suggest injecting new messages only if at least one virtual channel into each useful physical channel is free, or at least there is one physical channel with all their virtual channels free. An advantage of this method is that there is no threshold to adjust and it behaves quite well with different traffic patterns. The main drawback of the methods based on local information is that they work almost in "open-loop", getting slow and indirect feedback.

The latter ones, explicitly propagate congestion information that can be complex enough to indicate that congestion is starting in some zone of the network. Namely, in [6] a timeout-based method is described. The timeouts are used to measure how long a message head is stopped in the input buffers. If this time exceeds a threshold, congestion is assumed and some dedicated lines are used for propagating this information to the other nodes. A node that receives that signal reduces its injection rate. A drawback of this method is the adjustment of the timeouts. The technique proposed by Kim et al [7] allows the sender to kill any packet with a delay greater than a given threshold. In this approach, short packets are padded in order to assure that the sender can kill them before they reach their destination. This may produce network overload if messages are sent to distant nodes.

In [8], a mechanism that uses global information about the network is shown. In this case it takes a sole measure: the amount of occupied buffers in the whole network. In this way, the information about which nodes are responsible for the congestion, or which zones are more congested, is lost. The authors argue that this problem is solved with a mechanism for threshold self-tuning. They claim that with this mechanism, the thresholds reach different values depending on whether the congesting nodes are dispersed or grouped. However, this is just claimed, not shown in the paper. Moreover, some bandwidth is reserved to transmit this information. Bandwidth is a valuable resource and they should have analyzed what level of congestion the network would reach if the whole bandwidth had been used for sending user packets.

The main problem of the congestion control techniques based on global information lies in the resources needed for propagating such information to the nodes, leading to a traffic overload.

3 General approach

The general idea can be summarized as follows. First, detect congestion. Second, select the responsible node/s. Third, restrict message injection in some sources, by requiring the selected node to suspend injecting new messages into the network for a certain period of time. From here, a few questions need to be answered.

3.1 What is the criterion to detect the start of congestion?

The congestion detection can be carried out in each node by means of local or global information. In the local approach, a node detects congestion by itself, using its own buffers' occupancy, timeouts, etc. In the global approach, congestion is detected by using information about the whole network, which must be gathered from the other nodes. We are interested in the local approach because there is always a first node that detects the congestion locally.

Jain [9] already proposed to use the amount of packets in the input queues of routers, as a congestion indicator for the store-and-forward switching technique. Congestion is detected if there is more than one packet on average stored in the input queue. This measure is obtained by computing the average queue occupation for a given period.

Virtual cut-through offers the same advantages as wormhole at low load, namely, low message latency. However, it also offers the chance to use the fact that at high load, messages tend to occupy buffers in the same way as they do with store-and-forward.

Each buffer can store one or more messages. We can obtain a congestion indicator by taking the number of queued messages per physical channel. Initially, messages will not be queued, because they will be immediately routed as in wormhole. As congestion is appearing, messages tend to occupy the virtual channels buffers. In a few cycles, the amount of queued messages will cross a threshold and we will determine that congestion is present. Another possibility is to detect congestion when any buffer stores more than one packet.

Jain also suggests to calculate the mean queue length by sampling from the time that a queue becomes empty. However, in this particular context this might not happen, thus sampling the instantaeous queue lengths with a certain degree of skepticism is chosen. We are going to require the number of occupied buffers to exceed a threshold for a given period of time, in order to start to believe that congestion exists.

Summarizing, our approach consists of measuring the number of occupied buffers in each virtual channel of a node. This quantity is used to determine the presence of congestion in a local way. However, we can manage this indicator in several ways. For example, we can take the buffer occupancy in each routing, or the average buffer occupancy for a given period, or we can sample every given amount of cycles and decide that congestion is starting when that instantaneous value exceeds a threshold.

3.2 Which sources should be informed?

A first approach could be to act in the node that detected the congestion by limiting its message injection. However this can lead to an unfair situation, because while a node is throttling, its neighbours may be taking advantage of this, sending messages at the same rate without detecting congestion. Without enough care, some nodes might reach a state of starvation.

Therefore, it is very important to be able to estimate which nodes are responsible for the congestion. This can be done in several ways. One of them could be by propagating information about the congested zones and leaving each source to decide whether or not it must reduce its injection rate. Another one would be to use a received messages history and to extract from it the node that is sending more packets towards this zone. The chosen approach has been the latter.

3.3 How is injection controlled and by how much?

An easy method to control the injection of new messages is message throttling that delays in some cycles the injection of each new message. This reduces the applied load and allows the network to return to a normal working point.

We have decided to use global message throttling, that is, to apply message throttling in source nodes that are not necessarily the congestion detector. In this case, we need to inform about the congestion to that nodes.

However, to send congestion control packets through the normal lines when the network is in saturation is dangerous. Our control packet may arrive too late. A suitable solution is to use a virtual channel for control purposes. This virtual channel would have more priority that the rest and because of our network being based on virtual cut-through, this kind of control messages would arrive at its destination through a decongested path. It is important to note that in wormhole this technique would leave a fraction of our bandwidth not in use, but as we are using virtual cut-through, we can multiplex at packet level and do not waste bandwidth when there is no control packet to send. Nowadays, integrating a new buffer in our switch is not a problem.

Thus, when congestion is detected, a congestion control message will be sent to the node that we have estimated as responsible. When the source receives this control message, it delays in an initial number of cycles the injection of new messages. Next, the mechanism can behave by following two approaches,

1. If, after a given period of time, the switch continues detecting congestion, then it will send another

control message to the new responsible node (because now it may not be the same one). If a source that is limiting its injection rate does not receive a new congestion message, then it will reduce the delay until it reaches zero again. On the other hand, if it receives such a message, the delay will be increased as in the first case.

2. If, after a given period of time, the limited source has not received a "non-congestion" control message, it increases the message injection delay. This can be done following a linear or a geometric progression. When such a message is received, the delay is progressively decreased until it reaches zero again.

In order to reduce the number of control messages sent when the network is saturated, it would be preferable the second approach. However both of them have been tested to obtain a comparison.

3.4 How can we detect that congestion is about to disappear?

When the number of occupied buffers falls below the congestion threshold, we can affirm that congestion is disappearing. However, if there is only one threshold, the mechanism may be very sensitive to little interferences. Therefore, it may be convenient to use an upper and a lower threshold in order to have some hysteresys. However, due to the reasons exposed in subsec. 3.1, a sole threshold is used in this first approach. It may be necessary to check if the condition is kept for a certain number of cycles in order to believe it or to check it every given amount of cycles.

3.5 When and how can we restore the injection rate in the sources?

Let's assume that congestion is disappearing. Source nodes must act depending on the approach followed according to those described in subsec. 3.3. If the selected approach was the second one, each switch must remember at which nodes the congestion message was sent in order to send them the non congestion message. On the other hand, if the chosen approach was the first one, each source will stop limiting after a given period of time without receiving a new congestion message. The way in each source reduces its delay may be linear, geometric, logaritmic or a mixture of them. The linear way has been chosen for our first tests.

3.6 Is the congestion control mechanism fair?

A congestion control mechanism is said to be fair if only those nodes that are using more bandwidth than they should, react by limiting their injection rate. If this does not happen, we can find some network configuration with a good performance, but with unfair behaviour. For example, two nodes sending packets by a common path may be transmitting at very different rates. Our approach aims to reach an efficient and fair solution.

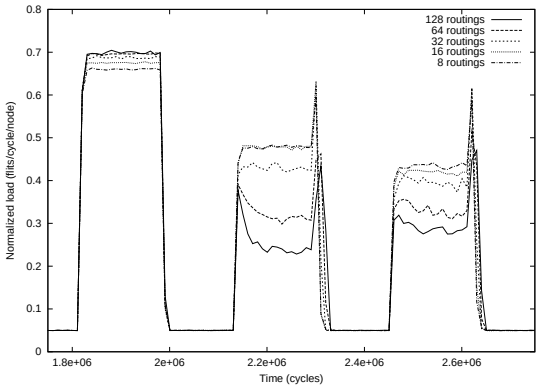

Figure 2. Delivered load vs. time when congestion is checked every 128, 64, 32, 16 and 8 routings. Message size of 64 bytes, timeout of 64 cycles in the source and 4 headers in the history. Uniform, bit-reversal and shuffle distributions.

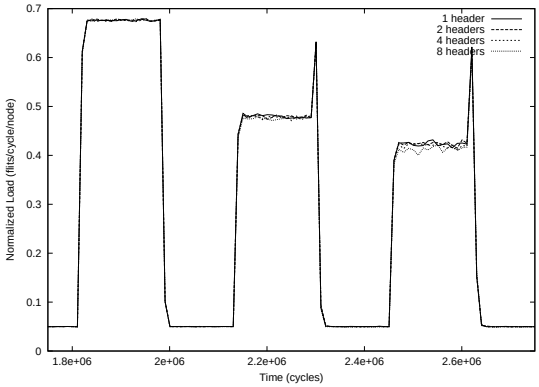

Figure 3. Delivered load with different sizes of history. Congestion is checked every 16 routings, message size of 64 bytes and timeout in source of 64 cycles. Uniform, bit-reversal and shuffle distributions.

4 Experimental results

We are interested in a global congestion control mechanism based on local detection. Our mechanism uses as a congestion estimator the number of occupied buffers in each virtual channel. A unique threshold with a value of two buffers, indicates when congestion is appearing and when it is disappearing. When the estimator exceeds the threshold, the headers of the buffered messages and a number of headers belonging to the

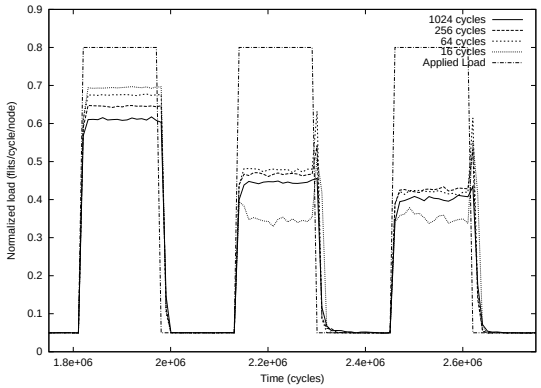

Figure 4. Delivered load with several timeouts. Congestion is checked every 16 routings, message size of 64 bytes and 4 headers in the history. Distributions: uniform, bit-reversal and shuffle.

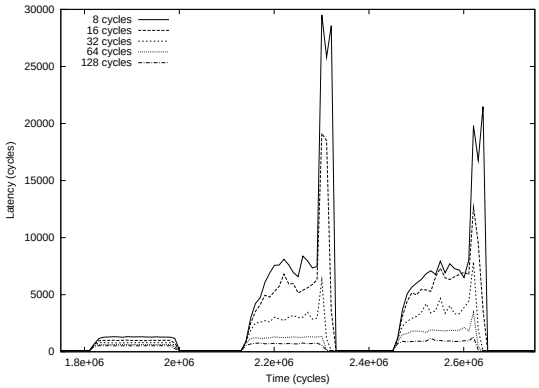

Figure 5. Latency vs. time with several timeouts, from 8 to 128 cycles. Distributions: uniform, bit-reversal and shuffle.

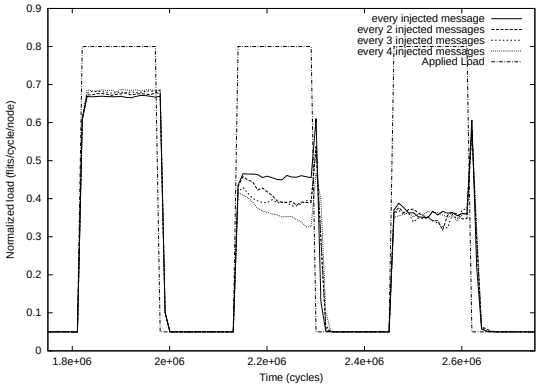

Figure 6. Load vs. time with the new approach.

most recently routed messages are checked. The most frequent source is taken as responsible for the congestion and a control message is sent to it.

Actually, in our simulations the congestion message is not sent to the source, the chosen source receives that information instantaneously. This is not the actual behaviour, but it provides an upper boundary of the performance of this method. The injection limitation is carried out by delaying in a number of cycles the injection of each new message. The load applied has been square pulses with a maximum of 0.8 flits/cycle/node and a minimum of 0.05 flits/cycle/node with message size of 64 bytes. The two approaches suggested in subsec. 3.3 have been used.

4.1 Our first implementation

Several simulations have been realized in order to show the behavior of the proposed method for congestion controlling. The network chosen was a k-ary n-cube with 512 nodes forming a 3D torus. The switching technique is virtual cut-through and there is only one host per switch. Three diferent destination distributions have been used: uniform, bit-reversal and shuffle. Every given period the virtual channels occupancy is checked in order to detect congestion. The injection is controlled by following the first approach shown in subsect. 3.3. The initial delay chosen was 32 cycles, and each new message adds 32 cycles to the delay. Every triggered timeout substracts 32 cycles.

There are several questions to be answered in this model.

- *How frequently must the router check the congestion estimator?* We have chosen to check the number of occupied buffers every given amount of routings because in this way it is easy to stop the routings for a while and then to send the control message if necessary. Figure 2 shows the network performance when different sampling frequencies are applied. The chosen frequency was every 16 routings.

- *What values must be assigned to the thresholds?* The value chosen has been two occupied buffers. The reason is that when the network is working normally, every received message is routed again without delay, occuping only one buffer. However it would be necessary to make a comparison among different threshold values.

- *How many headers must be stored in the history?* If the number is not big enough, the estimation of the responsible node could be erroneous and innocent nodes will be penalized. However, fig. 3 shows that the history size does not seem to be very important. This may be due to the fact that the headers of the stored packets are also used in the estimation.

- *What value of timeout is necessary to decide when to reduce the limitation?* This is a parameter which is difficult to adjust. Probably, it will de-

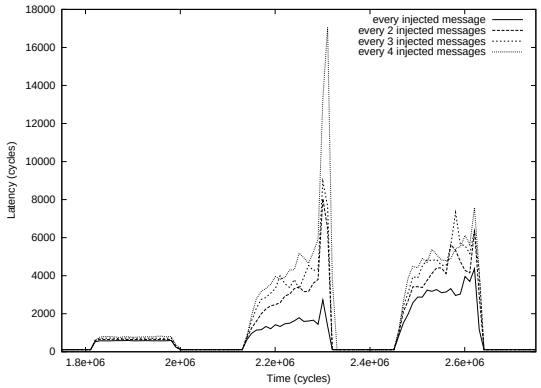

Figure 7. Latency vs. time with the new approach.

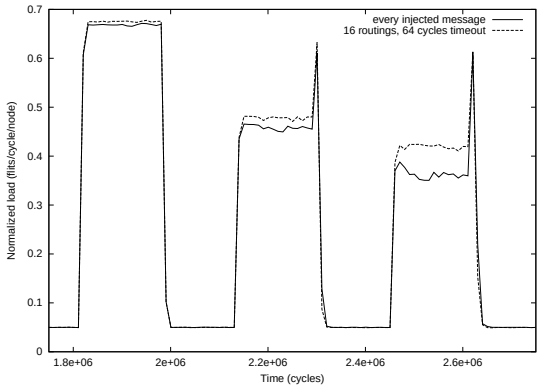

Figure 8. Comparison between the first and the second approach.

pend on the network size, the message length and the applied load. If we choose too low a value, the source will decrease the delay and after a few cycles it will receive a new congestion message, leading to the same delay as before. On the other hand, if the value is too big, the network will not react fast enough to adapt itself to the new situation, leading to delays in the messages. Figure 4 shows the delivered load versus simulation time with different timeouts.

Figure 5 shows message latency versus time with several timeouts. It can be noted that with a low timeout the latency behaves in the same way as it did without congestion control when the distribution is bit-reversal and shuffle. Uniform distribution has almost always good behavior.

5 Our second implementation

A drawback of our first implementation was the great number of parameters to adjust. In this implementation we have started from the same premises than our first one including the following changes.

The injection control mechanism is the second one proposed in subsec. 3.3. Every time a new message is received, the switch checks the congestion level. This is done by testing if the number of occupied buffers in that virtual channel exceeds two. In that case, a congestion control message is sent to the responsible node and its identifier is stored.

In each routing, the number of occupied buffers in the virtual channel is checked. If it is below two buffers, a "non-congestion" control message is sent to the responsible node, whose identifier was stored when the congestion was detected. If congestion is detected several times and the responsible node is different that the old one, a "non-congestion" control message is sent to the old source and a congestion one is sent to the new source of congestion. If, after a given number of injected messages, the limited source has not received a "non-congestion" control message, it increases the message injection delay. This is done following a linear progression, starting from 32 cycles and adding 32 cycles each time. When a "non-congestion" control message is received, the injection delay is progressively decreased in the same way until it reaches zero again. This approach has as an advantage that we must adjust just the number of injected messages before testing congestion in the source. Figures 6 and 7 show the admitted load and message latency respectively for different values of that parameter. As can be seen, the better performance is reached by increasing/decreasing the injection delay after every message.

6 Conclusions and Future Work

In this paper two new congestion control mechanisms have been proposed. Both of them are based on local detection of the congestion with message throttling in the sources that are estimated as causing the congestion. This estimation is made by means of a history containing the headers of the most recently sent messages. Figure 8 shows a comparison between the two approaches. The first method shows a better behaviour than the second one, however its main drawbacks are the bandwidth wasted in control messages and the difficulty in adjusting its parameters. The second method is easier to adjust but it offers a worse performance. Nevertheless, we have not taken into account the propagation delay of the control messages, hence obtaining an upper bound of the performance that each method can achieve. Both approaches show a good behaviour with uniform, bit-reversal and shuffe destination distributions.

Our future work includes: to reach a good adjustment of the parameters, to really send the congestion control messages in order to study the propagation effects in the performance and to use a more realistic distribution, perhaps by using real traces or autosimilar distributions.

References

[1] S. Warnakulasuriya and T. M. Pinkston, Characterization of deadlocks in interconnection networks, *Proceedings of the 11th International Parallel Processing Symposium (IPPS'97)*, April 1997, 80-86.

[2] L. Fernandez, J. M. Garcia and R. Casado, On Deadlock Frequency during Dynamic Reconfiguration in NOWs, *Proceeding of 7th International Euro-Par Conference*, Euro-Par 2001, Manchester UK, August 2001, LNCS 2150, 630-638

[3] E. Baydal, P. López and J. Duato, A Simple and Efficient Mechanism to Prevent Saturation in Wormhole Networks, *Proceedings 14th International Parallel and Distributed Processing Syimposium*, 2000, 617-622.

[4] P. Lopez, J. M. Martinez, J. Duato and F. Petrini, On the Reduction of Deadlock Frequency by Limiting Message Injection in Wormhole Networks, *Proceedings of Parallel Computer Routing and Communication Workshop*, June 1997.

[5] P. Lopez, J. M. Martinez, and J. Duato, DRIL: Dynamically Reduced Message Injection Limitation Mechanism for Wormhole Networks, *International Conference on Parallel Processing*, August 1998, 535-542.

[6] A. Smai and L. Thorelli, Global Reactive Congestion Control in Multicomputer Networks, *5th Int. Conf. on High Performance Computing*, 1998.

[7] J. H. Kim, Z. Liu, and A. A. Chien, Compressionless Routing: A Framework for Adaptive and Fault-Tolerant Routing, *Proceedings of the 21st International Symposium on Computer Architecture*, April 1994.

[8] M. Thottethodi, A. R. Lebeck, and S. S. Mukherjee, SelfTuned Congestion Control for Multiprocessor Networks, *Proceedings of the Seventh International Symposium on High Performance Computer Architecture (HPCA-7)*, January 2001.

[9] R. Jain and K. Ramakrishnan, Congestion Avoidance in Computer Networks with A Connectionless Network Layer: Concepts, Goals, and Methodology, *Proc. Computer Networking Symposium*, Washington D.C., April 11-13, 1988, 134-143.