Soft Real-Time Communication Over Dual Non-Real-Time Networks

(Extended Abstract)

Ben Kao* Hector Garcia-Molina†

November 23, 1992

Abstract

In this paper we consider systems with redundant communication paths, and show how applications can exploit the redundancy to improve real-time communications. We consider two policies, one that evenly distributes load, and one that partitions load according to packet slackness. We evaluate the effectiveness of these policies through analysis and simulation.

1 Introduction

With the development of distributed real-time systems, time constrained communication has become a topic of growing interest. The goal is to deliver packets at their destination by a given deadline, specified at packet arrival time. Applications that may require time constrained communication include packetized voice [6], distributed sensor networks [9], and real-time distributed databases [2]. Multimedia systems, integrating images, sound, and text, represent another important application. In this case, the communication system must support timely delivery of packets involved in real-time interactions, e.g., voice communication, tele-conferencing.

In this paper we study how *application level controls* can support *soft real-time communication traffic* on top of a *non-real-time network*. In particular, the objective will be to exploit

*Princeton University Department of Computer Science. Current address: Department of Computer Science, Stanford University, Stanford, CA 94305. e-mail: kao@cs.stanford.edu

†Stanford University Department of Computer Science. e-mail: hector@cs.stanford.edu

multiple redundant paths between communicating nodes. In the paragraphs that follow we motivate our interest in this particular problem.

It is clear that many hard real-time distributed applications (e.g., the space shuttle [7]) require specialized real-time networks that can guarantee timely delivery. At the same time, it is clear that the vast majority of the world's networks are not real-time, and that applications have little control over the order and time at which packets are transmitted. Standard networks such as Ethernet or token ring, and their corresponding drivers, simply do not support real-time traffic, and they tend to deliver messages in a FCFS order. Even for systems that provide priorities, they usually allow only a limited number of priority levels [13]. There also may not exist a simple mapping between the applications' sense of time criticality and the network's concept of priorities.

Given this state of affairs, can anything be done for all those applications running on non-real-time networks? Even though they are not hard real-time critical applications, it may be highly desirable to make them more responsive to the user timing requirements. For example, in a distributed database, it may be desirable to deliver messages involved in a two phase commit in a short time frame, to reduce the "window of vulnerability" where blocking can occur [5] and to reduce the time locks are held. Packets for read-only queries would have less stringent timing requirements. Packets for remote view update and quasi-copies [10] would have even less stringent requirements. Since most distributed databases will run on standard networks, how can the distributed database ensure that these timing requirements (or at least most of them) are met?

Given a single non-real-time standard network, where applications have no control over the low level transmission mechanism, there is not much that can be done to improve the system's real-time behaviour. However, if more than a single path or network is available to the application, then it may be possible to control the flow of packets to achieve real-time goals.

Fortunately, redundant paths or networks are not uncommon. Redundant networks are used to provide higher reliability and scalability. Redundant networks are used within a local area (e.g., a Tandem Nonstop system [1]), and in a wide-area environment by using different carriers or lines. Also, in order to exploit the potential of lightwave networks, it is often desirable to subdivide a high capacity channel into a multiple-channel network [3, 4]. In this paper we study how redundant paths or networks can be exploited to meet real-time communication

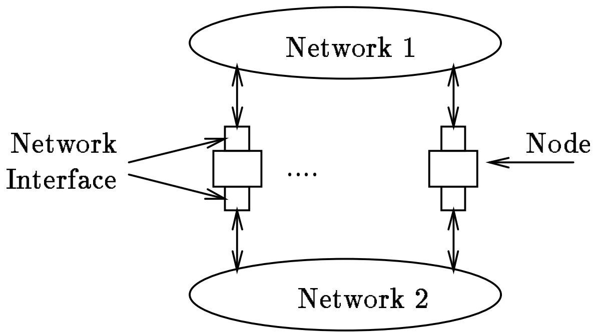

Figure 1: A Dual-network System.

constraints.

Note that since real-time networks are not common, their cost will be higher than standard ones, where economies of scale and simpler designs lower costs. Thus, it may be more cost effective to use redundant standard networks as opposed to a single specialized real-time network, achieving at the same time higher reliability and real-time responsiveness. This is another reason for investigating how time constrained communications can be carried over non-real-time networks.

For our evaluation, we will consider applications residing at a collection of nodes, each generating packets with real-time constraints. The nodes are connected by two independent networks (see Figure 1). Conceptually, the networks can be of any type. (However, in Section 4 we only simulate an Ethernet type network.) When a node has a packet to send, an application layer entity, which we will call a *dispatcher*, will choose a network through which the packet is to be transmitted.

We study and compare two schemes for dispatching packets over the two channels. The first scheme is to evenly distribute packets to the two channels. The second scheme is to prioritize packets into two classes depending on their time criticality. Packets with more stringent time constraints are routed to one of the networks, while the other network handles the less time-critical packets. The channel prioritization scheme is analogous to a four lane highway. The slower lanes on the outside carry more traffic (higher load); passengers on those lanes are in less of a hurry. The fast central lanes usually have fewer cars, traveling at a higher speed. In our networks, our goal will be the same as on the highway: the load on the network carrying the high priority packets will be kept lighter so they may meet their time constraints.

To analyze the relative performance of these dispatcher algorithms, we follow two strategies:

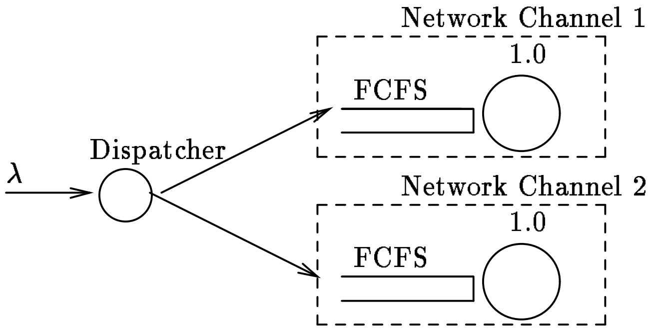

Figure 2: A Queueing Model.

First we construct a *very simple* queuing model that lets us study the major tradeoffs involved. Second, we construct a more detailed and realistic simulation model that lets us verify the trends observed in the analytical model in a specific setting. We stress that the analytical model is very simple. Its goal is not to predict performance of some real system; rather, its goal is to let us understand the key issues. As we will see, the model does give us much more insight than the more complex simulation model. Furthermore, the simulation does confirm the trends of the analysis (at least for a particular setting), so this gives us reason to believe that the analytical model is indeed capturing the key factors.

The rest of the paper is organized as follows. Section 2 presents the analytical model, Section 3 shows the analysis of a dual network with various dispatcher strategies. Section 4 presents the results of Ethernet simulations. Finally, we conclude the paper in Section 5.

2 The Model

Our analytical model is shown in Figure 2. We make the following assumptions: (1) The nodes in the system together generate a Poisson packet arrival stream, with mean interarrival time $1/\lambda$. (2) Packet transmission time is exponentially distributed with mean equal to 1 time unit. (3) Associated with each packet is its slack. We define slack to be equal to $t_s - t_g$ where t_s is the time at which the network must start transmitting the packet (to meet its deadline) and t_g is the generation time of the packet. For illustrative purpose, we assume slack is uniformly distribution over the range $[S_{min}, S_{max}]$. We define the symbol $k = S_{max} - S_{min}$. (4) Each network can be modeled by a FCFS server (M/M/1 queue).

As discussed in Section 1, this is a simple model. For example, actual networks may not process packets in a FIFO fashion. However, studies have shown that standard network protocols like Ethernet and token ring tend to queue packets in approximate FIFO order. Also, some access protocols (e.g. in Distributed Queue Dual Bus [12]) approximate a FCFS order of packet transmissions on a bus. Again, in our simulation we model a real Ethernet protocol, and show that the results are not that different from what the FIFO model predicts.

3 Dispatcher Strategies and Analysis

In this section, we consider a dual-network system with two possible dispatcher strategies and compare their performance.

Strategy 1 (Balance): Dispatch a packet to either channel with equal probability.

Strategy 2 (Chop): One network, N_T, is reserved for tight slack packets. It handles packets whose slacks are at the lower p quantiles of the slack distribution. Network N_L handles the rest of the load, i.e., packets with slack in the upper $(1-p)$ quantile of the distribution (looser slack).

The model of Figure 2 is evaluated using queueing theory and taking into account the load generated by each strategy, and the number of deadlines that are missed at each server. The details are given in [8].

To study the performance of the strategies, we select a set of representative base parameter settings ($\lambda = 0.8$, $S_{min} = 1$, $S_{max} = 15$) to illustrate the main tradeoffs involved. For the base setting we then vary one or two parameters at a time, showing their impact on performance. Here we only present some illustrative results; additional results are in [8].

Figure 3 shows the number of missed deadlines for the *Balance* and *Chop* strategies ($MD_{Balance}$, MD_{Chop}), as a function of p. $MD_{Balance}$, does not vary with p and is a constant 2.6%. We see that the performance of *Chop* is sensitive to the choice of p. If p is chosen too small (< 0.24), network N_L is congested with a load (> 0.76) that it cannot properly handle. Strategy *Chop* is counterproductive in this case. However, over a wide range of p values, [0.24,0.5] *Chop* does outperform *Balance*. Also, observe that the curve is quite flat near the region of the optimal p value. Therefore, a small variation in p from the optimal point does not have serious consequences. Finally, we note that the optimal value of MD_{chop}, $MD_{chop}^{opt} = 1.8\%$ occurs at $p = 0.35$. This is about 30% fewer missed deadlines than strategy *Balance*, a significant improvement.

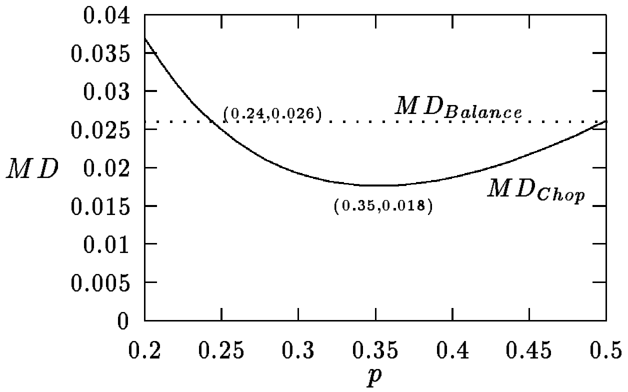

Figure 3: MD_{Chop} as a function of p. $\lambda = 0.8$, $S_{min} = 1$, $S_{max} = 15$.

To study the impact of load on performance, we vary the load λ from 0.2 to 1.0, (Note that since we have 2 networks, we can have loads greater than 1.0 without saturating the system.) Figure 4(a) shows $MD_{Balance}$ and MD^{opt}_{Chop} as λ varies. (For a given λ, MD^{opt}_{Chop} is obtained by selecting the p that minimizes the fraction of missed deadlines.) We see that when λ is small, MD^{opt}_{Chop} increases at a slower rate than $MD_{Balance}$. It then slowly catches up as λ becomes larger. As the load grows further to a high value, both strategies perform equally poorly. (The curve $MD^{0.9opt}_{Chop}$ is discussed below.)

Figure 4(b) shows the relative gain in switching from *Balance* to *Chop* by presenting the ratio $MD_{Balance}/MD^{opt}_{Chop}$. We see that *Chop* performs much better than *Balance* under low load. (Better by a factor of 2). As a matter of fact, one would expect the load to be quite low in any system with real-time deadlines, else too many deadlines would be missed. Thus, the *Chop* has its most impressive gains in the most likely operating range.

As pointed out earlier, MD_{Chop} is sensitive to the choice of p. But how bad is it if we choose the wrong p? In Figure 4, we show MD_{Chop} evaluated at a suboptimal $p = 0.9 \cdot p_{opt}$, where p_{opt} is the optimal p value. The resulting curve is labeled $MD^{0.9opt}_{Chop}$ in Figure 4(a). We observe that under low load, the fraction of missed deadlines hardly changes as a result of the variation in p. As the load increases beyond 1.0, the variation in p is significant, and *Chop* performs as poorly as *Balance*. Again, we note that a real-time system should have considerable spare capacity, so that the vast majority of deadlines can be met. Therefore, under normal condition, the system

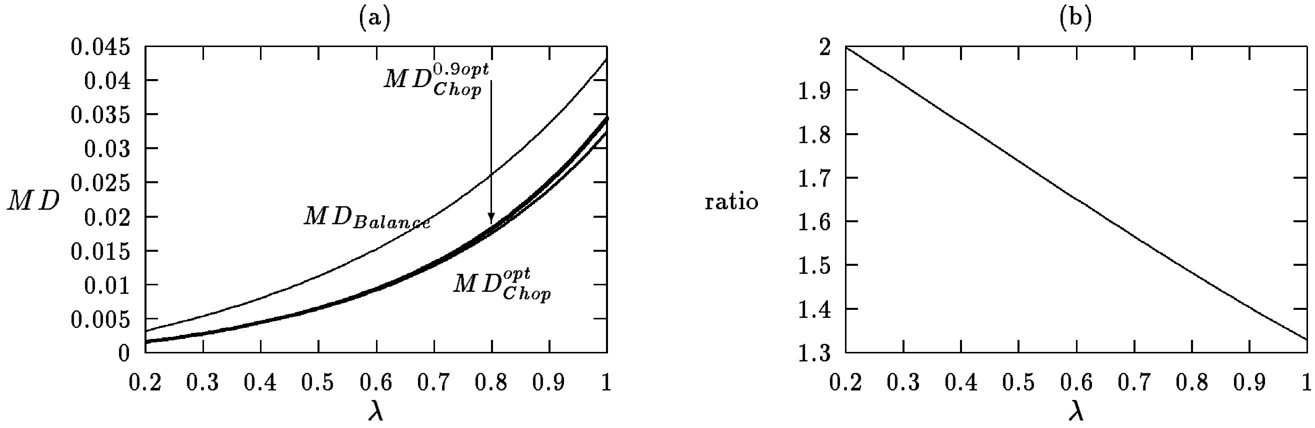

Figure 4: $MD_{Balance}$, MD_{Chop} and $\frac{MD_{Balance}}{MD_{Chop}}$ as λ varies.

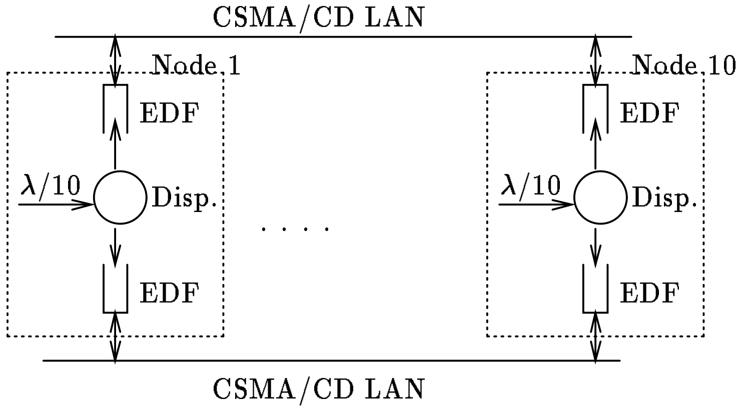

Figure 5: Ethernet Simulation Model. (EDF – Earliest Deadline First Queue)

should not be operating at a high load. This suggests that inaccuracy in finding an optimal p should not diminish the advantages of $Chop$.

In conclusion, our simple analytical model indicates that $Chop$ outperforms $Balance$, as long as a reasonable p value can be selected. The gain can be very significant if either the system load is low or there is a wide variation in packet slackness [8].

4 Ethernet Simulation

To study the $Chop$ strategy in a realistic scenario, we implemented a detailed event-driven simulation of a dual Ethernet system. Instead of modeling each network by a FCFS server with a global queue, we simulate 10-node (or station) Ethernet (See Figure 5). Each node

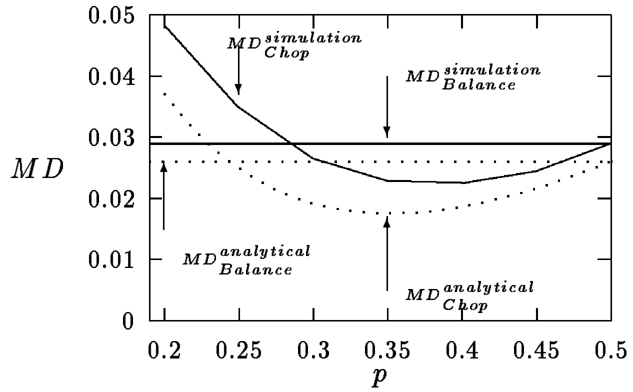

Figure 6: MD_{Chop} as a function of p. $\lambda = 0.8$, $S_{min} = 1$, $S_{max} = 15$. Ethernet Simulation.

generates a Poisson packet stream with mean rate $\lambda/10$ (total load thus equals λ). Two buffer queues at each node, one for each network, are used to internally store packets waiting for service. The queueing discipline is non-preemptive earliest deadline first (EDF). 1-persistent CSMA/CD with binary exponential backoff is implemented as the network access protocol [11]. The transmission time is still assumed to be exponentially distributed with mean equal to 1 time unit. Finally, the round-trip latency of the network is taken to be 1% of the mean packet transmission time (i.e. 0.01 time unit[1]).

Each simulation run lasts 200,000 time units. Each data point is obtained by averaging the results of 5 simulation runs, each with a different random number seed. The 90% confidence interval is $\pm$ 4% of the value shown, or about $\pm$ 0.002.

The solid lines in Figure 6 show the simulation results comparing the *Balance* and *Chop* strategies as p varies. The curves in Figure 3 (page 6) are reproduced in Figure 6 as dotted lines. These dotted lines show the results as predicted by the analytical model.

Comparing the simulation results (solid lines) and the analytical results (dotted lines), we observe that the analytical model underestimates the missed deadlines MD when Ethernets (Figure 5) rather than FCFS servers with global queues (Figure 2) are assumed. We also note that the underestimation decreases as p approaches 0.5.

This underestimation is attributable to the additional delay introduced by network con-

[1]Based on a 500m 10 Mbps Ethernet cable with a propagation speed of 4.33μsec/km, and 512 bytes packet size.

tention. Time is wasted in packet collisions. Furthermore, a packet may be delayed because of multiple conflicts, due to the probabilistic nature of the access protocol. This effect translates into a higher MD value than predicted. Moreover, when p is small (e.g. 0.2), the low priority network is given a high load (e.g. 0.8). The degree of contention in the low priority network is thus high, making the difference between the analytical MD value and the simulation MD value larger.

Notwithstanding the discrepancy, Figure 6 shows that both the simulation and analytical results display similar trends. In particular, (a) *Chop* performs better than *Balance* over a significant range of p values, (b) the slope of MD_{Chop} is relatively flat at the optimal point, so that a minor deviation from the optimal value can be tolerated, and (c) the gain by using *Chop* over *Balance* is significant.

We performed extensive experiments with the simulator, varying parameters over wide ranges (including different numbers of nodes on the network), and comparing the results to those of the analytical model. In all cases, the simulation model confirms the trends identified by the analytical model. Thus, we conclude that the simple analytical model is useful for understanding dual networks and real-time communication.

5 Conclusion

In this paper we have shown how multiple non-real-time networks can be used to support soft real-time communication. Our approach is unusual (at least from the perspective of conventional real time systems) because we are assuming that (1) we cannot afford or do not have available a real-time network; and (2) we can tolerate some missed deadlines (soft real-time systems). We believe there is a wide class of applications where this is true, and where strategies like *Chop* can significantly reduce the number of deadlines that are missed.

As has been pointed out earlier, the performance of the *Chop* strategy depends on the p values used by the dispatchers. Although it is not crucial to have exactly the optimal p value for a given configuration (see, for example, Figure 3), it is important to have a reasonable value. For example, if the system parameters are static, the analytic model could be used to select the optimal p, which our simulation results show to be close to the actual optimal p. If the system is dynamically changing, an adaptive strategy that automatically tunes the p choice can be used. Readers are referred to [8] for more details on this issue.

It is interesting to note that our basic idea, that of splitting load according to real-time slackness, can be applied to other replicated system components. For example, in a database system with replicated databases, read requests could be routed according to transaction slack. Similarly, with mirrored disks, IO requests could be split, or in a multiprocessor system, requests could be queued by slackness.

References

[1] *Introduction to NonStop Systems, 16270.* Tandom Computers Inc. Cupertino, CA, 1991.

[2] R. Abbott and H. Garcia-Molina. What is a real-time database system? In *Abstracts of the Fourth Workshop on Real-time Operating Systems, IEEE Computer Society*, pages 134–138, 1987.

[3] A. S. Acampora. A multichannel multihop local lightwave network. In *Proceedings of the IEEE GLOBECOM*, pages 1459–1467, 1987.

[4] W. Bandula, Abeysundara, and Ahmed E. Kamal. High-speed local area network and their performance : A survey. *ACM Computing Surveys*, 23(2):221–264, June 1991.

[5] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

[6] T. Bially, A.J. Mclaughlin, and C. J. Weinstein. Voice communications in integrated digital voice and data networks. *IEEE Trans. Communications*, 28(3), march 1980.

[7] G. D. Carlow. Architecture of the space shuttle primary avionics software system. *Communication of the ACM*, 27(9), sept 1984.

[8] B. Kao and H. Garcia-Molina. Real time communication over multiple standard networks. Technical Report STAN-CS-92-1453, Stanford University, 1992.

[9] MIT Lincoln Laboratories. *Workshop on Distributed Sensor Networks.* 1982.

[10] B. Lindsay, L. Haas, C. Mohan, H. Pirahesh, and P. Wilms. A snapshot differential refresh algorithm. In *Proceedings of the ACM SIGMOD*, pages 53–60, 1986.

[11] R. M. Metcalfe and D. R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404, July 1976.

[12] R. M. Newman and J. L. Hullet. Distributed queueing: A fast and efficient packet access protocol for QPSX. In *Proceedings of the 8th Internatinoal Conference on Computer Communications*, pages 294–299, 1986.

[13] L. Sha and J. P. Lehoczky. Performance of real-time bus scheduling algorithms. *ACM Performance Evaluation Review, Special Issue*, 14(1):44–55, 1986.