Self-Synchronization Phenomena in Computer Networks
6.892 Project Report

Win Treese

16 December 1992

1 Introdution

There are many anecdotes in the computer networking community describing inadvertent synchronization of systems on a network. Until recently, this phenomenon was not investigated in detail, although some authors allude to the problem, as in the standard for the Routing Information Protocol[4]. In most cases, once the problem was identified, the algorithms were modified to avoid synchronization.

This paper considers a particular model of certain network algorithms that can result in synchronization. We present results from simulation, experiment, and theoretical analysis of the model. From the simulation results and analysis, we can make some general statements that characterize the conditions under which synchronization may occur and how algorithms may be modified to avoid synchronization.

The analysis of synchronization is not new in other fields, such as physics and biology. Christian Huygens, who published the first known description of synchronization in the seventeenth century, observed that a pair of pendulum clocks suspended from a beam became synchronized, even though they ran at different rates when separated. Since that time, synchronization has been discovered, analyzed, and sometimes put to use in a number of fields, including the design of electric generators. Many of these are discussed in [1]. Synchronization plays an important role in biological systems[6], at scales ranging from small groups of cells to the behavior of a collection of individuals.

After beginning the work described here, I became aware of work in progress by Sally Floyd and Van Jacobson at Lawrence Berkeley Labs[3]; the presentation here is heavily influenced by that work.

2 The Algorithm

Following [3], we consider a model of Periodic Messages. In this model, each participating system periodically broadcasts a message to all other participating systems. The systems behave as follows:

1. Set a timer for T_p seconds in the future
2. Actual wakeup occurs at a time drawn uniformly from the interval $[T_p - T_r, T_p + T_r]$, where T_r is a parameter of the system that bounds the random variation in the wakeup interval.
3. Prepare and broadcast a message to all other participating systems. This requires T_c seconds of time.
4. Process any messages that have arrived during the awakened time, including the time taken processing these messages. Each message takes T_c seconds to process.
5. Go to to step 1.

Any messages that arrive during the "sleep" period are processed immediately.

This algorithm is an abstraction of several algorithms that have been implemented for network routers and other network protocols. It is easy to see why such an algorithm might be implemented: to avoid the overhead of scheduling and switching contexts, messages are processed while the manager is already running.

3 Analytical Models

Of the many models that have been analyzed for large collections of interacting oscillators, the simplest[2] is

$$\dot{\theta}_j = \Omega_j + \sum_{i \in I_j} h_{ij}(\theta_i - \theta_j, \epsilon)$$

where θ_j is the phase of the jth oscillator. The set of neighbors for the jth oscillator is denoted by I_j, h_{ij} describes the coupling between pairs of oscillators, and ϵ is a parameter representing the strength of the coupling. Models of this form are often used to describe biological systems, and they have been specialized to analyze oscillators that are coupled to a set of nearest neighbors (e.g., linear, ring, mesh), for pools of oscillators in which each oscillator affects all other oscillators, and a variety of other similar systems.

From the abstract algorithm described in section 2, we can derive the following equation describing the behavior of the systems:

$$\dot{\theta}_j = \bar{\omega} + R_i(t) + T_c \sum_i h(\theta_i - \theta_j)$$

where:

$$h(x) = \begin{cases} 0 & \text{if } |x| \geq T_c \\ 1 & \text{if } |x| < T_c \end{cases}$$

and $\bar{\omega}$ is the nominal frequency, which is the same for all of the oscillators. $R_i(t)$ is a random value drawn from the interval $[-T_r, T_r]$; a new value is drawn every time time step. Note that the coupling function $h(\theta_i - \theta_j)$ only has an effect when two systems broadcast within T_c seconds of each other. This corresponds to step 4 in the algorithm, where incoming messages are processed before resetting the timer.

Some qualitative aspects of this model are immediately apparent. First, if we begin with N unsynchronized systems, then the change in phase is dependent entirely on the random component $R_i(t)$, and each system will be exhibiting a random walk about its initial phase.

If two systems wander within a distance T_c of each other, then the coupling effect will contribute to a phase advance. Because $R_i(t)$ continues to affect the phase, it is possible that two synchronized systems will become unsynchronized. As long as they remain synchronized, however, the phase advance will cause this "cluster" of systems to "catch up" in phase with other, unsynchronized systems. When this occurs, the larger cluster will sweep forward even more rapidly. It is likely that once this process begins, all systems will eventually be pulled into the cluster. The details of the evolution of the system, then, depends largely on the random component $R_i(t)$, which affects both how easily clusters form and how easily they break up. A small magnitude would suggest that clusters are less likely to form, but are also less likely to break up once they do form. A large magnitude, on the other hand, implies that clusters may form easily, but break up easily as well.

4 Simulation

In order to investigate the behavior of this model, I implemented a simulator in the Scheme programming language (the code for the simulator is included in appendix A). The simulation results agree qualitatively with those suggested by the analytic model, and with the simulation results reported in [3].

The parameters used in this simulation, as in [3], were $T_p = 121$ seconds and $T_c = 0.11$ seconds. The maximum magnitude of the randomness parameter T_r was varied over the runs of the simulation. The figures are plotted with time on the horizontal axis and the "phase" on the vertical axis taken as the wakeup time of the system modulo $T_p + T_c$. Note that the time scale on the horizontal axis varies from figure to figure in order to better show the salient details.

Figure 1 shows one run of the simulation for $T_r = 0.08$ seconds. In this case we see several clusters form; one grows enough that it eventually sweeps all nodes into the cluster. Other clusters may break apart, before the nodes are later claimed by the large cluster. This is as we expected, because of the phase advance caused by the coupling.

Another run of the simulation (figure 2) with the same value of T_r demonstrates that synchronization does not always occur quickly for this value.

If we take T_r much smaller, there is little variation in phase, unless two systems become synchronized. Those synchronized systems, however, are unlikely to come apart, and sweep the others, as shown in figure 3, where $T_r = 0.04$ seconds.

Finally, if T_r is larger than T_c, the clusters that form are unstable, because of the relativly large variation. This is demonstrated in figure 4, where $T_r = 0.12$ seconds.

The results of the simulation confirm the qualitative expectations from the analytic model, and they agree in general with the behavior of real systems that exhibit synchronization.

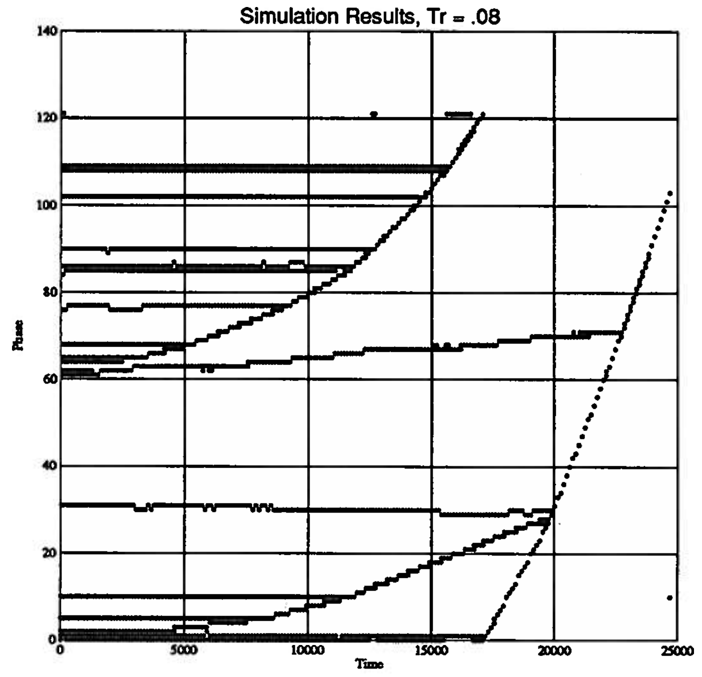

Figure 1: $T_r = 0.08$ seconds

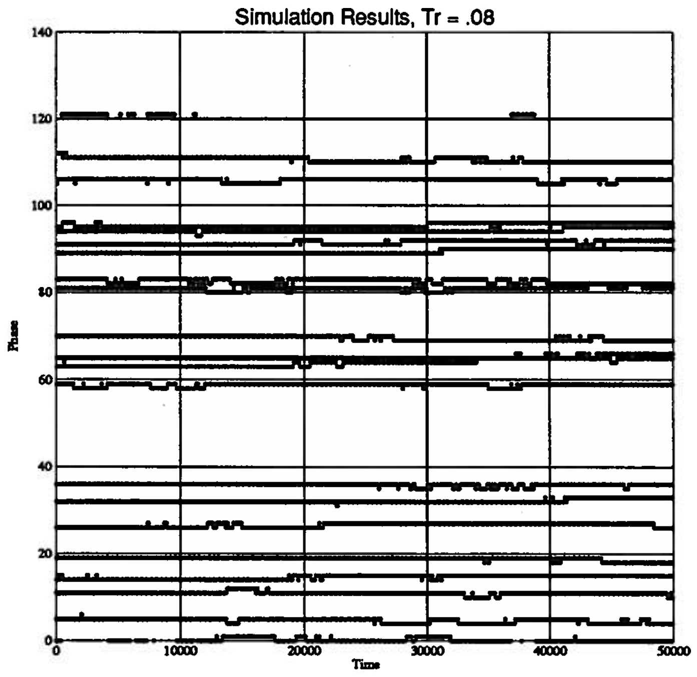

Figure 2: $T_r = 0.08$ seconds

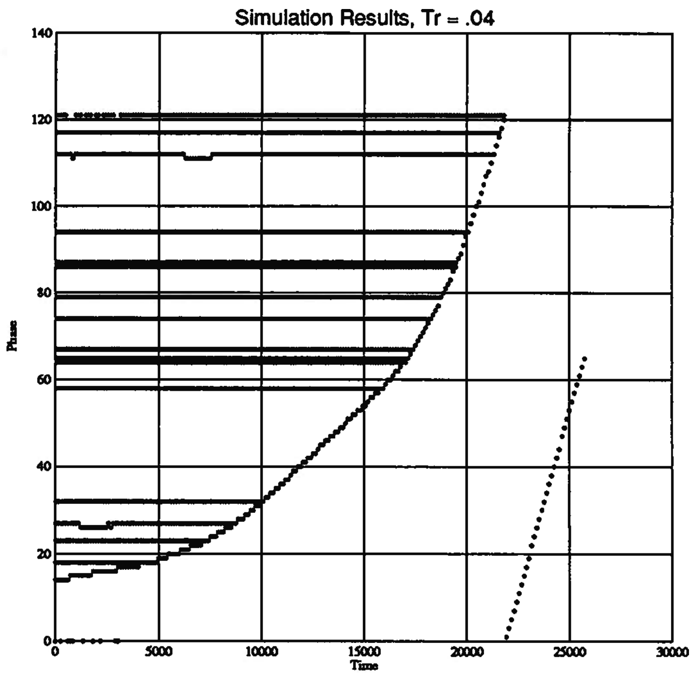

Figure 3: $T_r = 0.04$ seconds

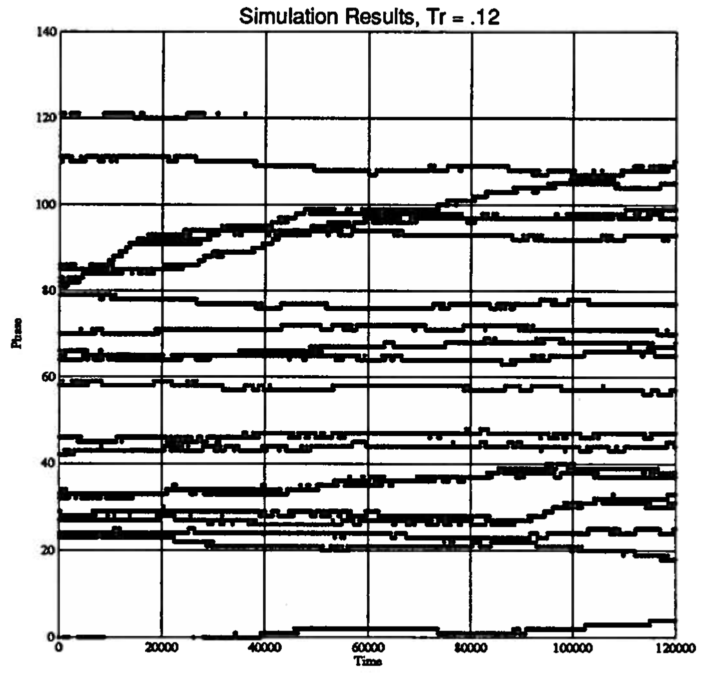

Figure 4: $T_r = 0.12$ seconds

5 Experimental Results

In addition to the simulation, I implemented the algorithm described above to run on a set of workstations on an Ethernet. At this time, a few short runs of this program has not demonstrated strong synchronization behavior. Some work remains to ensure that the implementation correctly implements the model, however, and the runs should extend over a longer period of time.

6 Discussion

The algorithm presented here is an abstraction of the algorithms that might be used in designing a standalone network router. There are two obvious techniques for avoiding the synchronization problem. First, the timer could be reset immediately upon wakeup, rather than after the processing. This is a particularly efffective technique, because it removes the coupling from the system entirely. In some systems, however, this change may be difficult, or it may cause a degradation in performance. An alternative is to inject some randomness into the timer. For the model described here, this has the effect of increasing T_r. From the simulations, it is clear that a sufficiently large value for T_r relative to T_c will make synchronization unlikely. On the other hand, an insufficient T_r may encourage synchronization.

While the model of Periodic Messages as described above captures the essence of the synchronization phenomenon in this system, there are some issues in using the results for implementations. For example, the noise component, bounded here by T_r, is likely to be drawn from a normal distribution rather than a uniform one. This makes it harder to be give good rules for the relationship between T_r and T_c to avoid synchronization. In addition, many real systems will not awaken a task before the requested time, so that phase advances are common, but phase delays are not.

7 Conclusions

We have described a simple network algorithm that shows a strong tendency towards global synchronization for certain parameter values. Variations of this algorithm have been implemented for several different systems, notably network routers. With a better understanding of this phenomenon, it is straightforward to design algorithms that do not suffer from this undesired synchronization.

Network algorithms with a strong periodic component are relatively unusual in the Internet community now. There is, however, increasing use of multimedia systems on networks – the audio and video components of which have do have strong periodic components. From the analysis presented here, it is unclear how likely synchronization problems with such systems will be. In particular, the coupling mechanism is somewhat different, so this problem remains for future work. One possible avenue to explore is the problem of nearest-neighbor couplings for period traffic through a wide-area network. A general analysis of this problem can be found in [5].

Another interesting question is whether or not it may be possible to make profitable use of such synchronization phenomena. It seems unlikely that this kind of system would be useful for clock synchronization, especially compared to other known algorithms. On the other hand, it may be that there are applications for which this kind of synchronization is beneficial. One possible avenue of work is to take advantage of phase-locking behavior to lock periodic processes with different phases, so the effects on the network are spread out in time.

Finally, there may be further applications of the theory of dynamical systems to the analysis of the local and global behaviors of computer networks. Such an analysis might provide a framework for analyzing algorithms for congestion control and avoidance, flow control, etc.

8 Acknowledgements

Thanks to Dave Clark for suggesting this problem.

References

[1] I. I. Blekhman. *Synchronization in Science and Technology.* ASME Press Translations, 1988. translated from the Russian by Eugene I. Rivin. Russian edition published by Nauka Publishers in 1981.

[2] Hiroaki Daido. Lower critical dimension for populations of oscillators with randomly distributed frequencies: A renormalization group analysis. *Physical Review Letters*, 61(2):231–234, 1988.

[3] Sally Floyd and Van Jacobson. The synchronization of periodic routing messages. Unpublished Draft.

[4] Charles Hedrick. Routing information protocol. Internet Request for Comments 1058, June 1988.

[5] Steven H. Strogatz and Renato E. Mirollo. Phase-locking and critical phenomena in lattices of couple nonlinear oscillators with random intrinsic frequences. *Physica D*, 31:143–168, 1988.

[6] Arthur T. Winfree. *The Geometry of Biological Time.* Springer-Verlag, 1980.

A Simulation Code

This appendix contains the Scheme source code for the simulation.

```
(define Tp 121.0) ; Nominal period in seconds
(define Tc 0.11) ; Computation time (seconds)
(define Tr 0.08) ; Random variation in period (seconds)
(define T (+ Tp Tc)) ; Time offset

;; A sender is represented by an event-time.  If it is on the
;; wait-queue, the time represents when it will wake to send its
;; message. If it is on the busy queue, the time represents when
;; it will wake up and be done.

; Generate a list of N initial times in the interval [0, Tp]

(define (make-simulation n)
  (if (= n 0)
      '()
      (cons (* Tp (get-random))
    (make-simulation (- n 1)))))

; Top-level: run a simulation with N systems

(define (run-sim n)
  (next-step (sort (make-simulation n) <) '())))

(define *BIG* (exact->inexact (expt 2 20)))

(define (next-step wait busy)
  ; Pick next event off of one of the queues
  (let ((next-wait (if (null? wait) *BIG* (car wait)))
(next-busy (if (null? busy) *BIG* (car busy))))
    (if (< next-wait next-busy)
  (let ((x (int next-wait))
(y (mod-T next-wait)))
    (graphics-draw-point win x y)
    (next-step (cdr wait)
       (if (null? busy)
   (list (+ next-wait Tc))
   (map (lambda (x) (+ x Tc)) (cons next-busy busy)))))
  (next-step (sort (cons (next-time next-busy) wait) <)
    (cdr busy)))))


(define (mod-T t1)
  (let ((x (/ t1 T)))
    (inexact->exact (round (* T (- x (truncate x)))))))

(define (next-time t1)
  (+ t1 (- Tp Tr) (* (get-random) 2 Tr)))

;; Utility procedures

; get a random fraction between 0 and 1, with resolution res

(define (get-random)
  (define res 10000)
  (exact->inexact (/ (random res) res)))

(define (int x)
  (inexact->exact (round x)))
```