INTERNETWORKING: RESEARCH AND EXPERIENCE, VOL. 1, 101–112 (1990)

# Oscillating Behavior of Network Traffic: A Case Study Simulation

LIXIA ZHANG
*Palo Alto Research Center, Xerox Corporation, 3333 Coyote Hill Road, Palo Alto CA 94304, U.S.A.*
DAVID D. CLARK
*MIT Lab for Computer Science, Cambridge, MA 02139, U.S.A.*

**SUMMARY**

**The main purpose of this paper is to document data traffic oscillation phenomena that have been observed both in operational networks and in simulation. In particular, by means of simulating TCP/IP network operations we are able to examine in detail the causes of traffic oscillation in a simple network setting. Our analysis shows that users' control actions are highly synchronized by the network congestion signaling and that providing users with only a binary network state can lead to repeated traffic oscillation.**

**It is yet to be determined by future work, however, to what extent the observed oscillation is due to the specific congestion control algorithm being used, and to what extent oscillation is an intrinsic phenomenon in large-scale, distributed systems and thus unavoidable. We suggest providing users with selective feedback and more quantitative information about network state as one way to improve traffic stability; the plausibility of this suggestion must be verified through simulation and implementation.**

KEY WORDS Network congestion Stability

## 1. INTRODUCTION

In packet-switched networks, it is highly desirable to maintain a stable traffic flow to avoid switch buffer overflow while maintaining high resource utilization, and to keep a stable end-to-end transmission delay (which can in turn make superfluous retransmissions unlikely to occur). Network traffic oscillation has been measured both in operational networks and in simulation. Oscillating traffic can lead to the following undesirable consequences:

1. At peak traffic resonance, switch buffers overflow and some packets are discarded. Consequently, retransmissions are used to recover the losses, which result in out of order packet arrivals at the receiving end.
2. Network resources are used unevenly; they are wasted both by retransmissions and by the idle time between oscillations.

1049–8915/90/020101–12$06.00

*Received February 1990*
*Revised June 1990*

3. Worst of all, end users experience long and highly varying network transmission delays (see the measured end-to-end delay curves in Figure 4). The delay makes the network incapable of serving real-time (e.g. packet voice) or interactive (e.g. remote log-in) applications.

Therefore it is important to understand the underlying causes of network oscillation.

Jacobson (1987) was the first to observe oscillatory behavior in ARPANET traffic. He sent ICMP echo messages (Postel, 1981b), at a constant rate, to a specific destination host across the ARPANET, and then measured the round-trip time (RTT) of each echo. He plotted a curve of the measured RTTs versus time which shows a periodic oscillation.

In our network simulation study, a similar oscillatory phenomenon was also observed (Hashem, 1989; Zhang, 1989). We conducted network simulations to study the performance of datagram networks with end-to-end window flow control mechanisms. The TCP/IP protocol suite (Postel, 1981a,c) was chosen as an ideal candidate for this purpose. It is well-known and widely deployed. In addition, the recently developed congestion control algorithm, Slow-Start (Jacobson, 1988), has further enhanced the protocol and brought significant performance improvement over previous TCP/IP implementations.[1] The TCP/IP implementation mentioned in the rest of this paper is assumed to be the version with the Slow-Start enhancement.

One of the most significant observations from our simulations of TCP/IP network operation is wild oscillation of packet flows. Periodically, packet queues at network bottleneck points grow to the buffer capacity and then shrink to nothing. In this paper we examine in detail the causes of this traffic oscillation phenomenon by simulation of a simple network setting. We discovered that, surprising as may it sound, the flow control adjustments of individual TCP connections are highly synchronized, and it is this synchronization that leads to the network traffic oscillation. The synchronization is in turn due to the fact that all the end users receive network congestion signals simultaneously. Consequently, the users slow down, and then speed up their data transmission in a synchronized manner, making the network state repeat the cycle of being congested–empty–congested again indefinitely.

We present the simulation model in Section 2 and the traffic oscillation observed in Section 3. Simulation tools allow us to trace detailed protocol operations to explore the causes of oscillation in Section 4. We then discuss our results as compared to other work in this area. The last section summarizes our current discoveries and proposes directions for future research effort.

## 2. SIMULATION MODEL

This section describes the network model used in our TCP/IP network simulation experiments. Since the TCP/IP protocol suite is well-known, we omit the protocol description here and refer interested readers to Postel (1981a,c) for details. We briefly describe the Slow-Start congestion-control algorithm below.

[1] A number of TCP users reported throughput improvement by a factor of 2 to 5 in the operational Internet (TCP-IP mail communications).

## 2.1. Slow-Start algorithm

TCP runs on top of IP, a datagram protocol that provides essentially no feedback information about the network status. Early implementations of TCP naturally chose a fixed window size (specified by the data receiving end) as default for all connections. The Slow-Start algorithm (Jacobson, 1988) enhances TCP's flow control with a dynamically adjusted *congestion control window* and uses the acknowledgment return and the retransmission time-out as control information. The window size used to regulate data transmission is

min(receiver window, congestion control window)

Packet losses, which are approximated by retransmission time-outs, are used as a congestion signal and acknowledgment return as an indication of an uncongested network. In the Slow-Start algorithm, each TCP connection starts with a congestion control window size of one maximum-size packet, and opens the window gradually upon receiving acknowledgments. When a retransmission time-out occurs, the control window closes down to one packet; it then reopens gradually upon further acknowledgment returns. To reduce throughput losses during the period of small control window size, the window opening is divided into an exponential opening phase and a linear opening phase. This means a faster opening at the beginning and a slower increase later when the window size is getting close to the point at which congestion was last experienced. See (Jacobson, 1988) for more details.

## 2.2. Network model

Two topology models are used in simulation: a simple network with one bottleneck link and a network with four switches in a row (see Figure 1).

All the links between hosts and switches have a bandwidth of 10 Mbit/s and a negligible propagation delay, to mimic commonly seen high-speed LAN connections from hosts to local gateways. The links between switches have a bandwidth of 100 kbit/s and a propagation delay of 5 ms, approximating the condition of some of today's long-haul network connections. The switches in Topology-1 have a buffer size of 30 packets each, and the switches in Topology-2 have a buffer size of 100 packets each. All the network

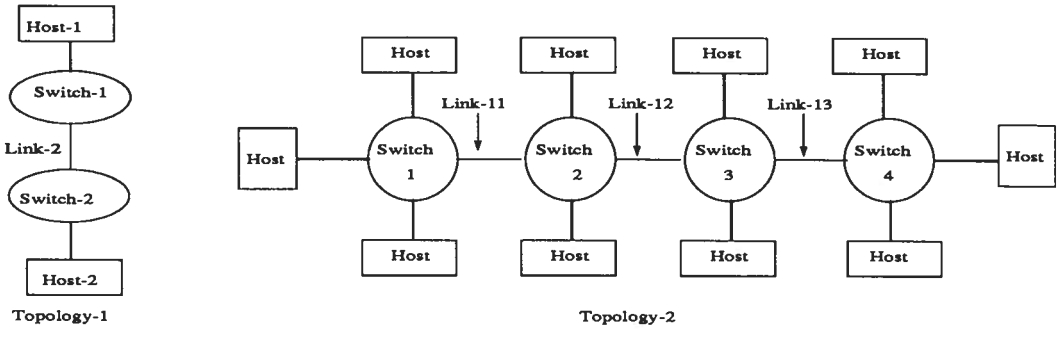


*Figure 1. The two network topologies used in simulation tests*

links and switches are assumed to provide error-free transmission. Because the Slow-Start congestion-control algorithm uses packet losses (which can result from either congestion or bit errors) as the congestion signal, this assumption eliminates the problem of false congestion alarms resulting from transmission errors in simulation. However, such false alarms present a serious problem in a real network environment (Seo *et al.*, 1988).

### 2.3. Load model

Data traffic in the simulation is generated by TCP connections. In the rest of the paper the word *user* refers to the end-to-end TCP connections. All the simulation tests mentioned in this paper are based on the following assumptions:

1. Each TCP connection starts at a random time and keeps running forever.
2. Each connection has an infinite amount of data to transmit, hence the data flow is restricted only by the protocol's flow control mechanism.[2]
3. Each receiver specifies a window size of 20 packets. If this window size is too small, it will limit the connection's throughput; if it is large enough such that the data transmission is controlled solely by the congestion control window, then the exact value is not important.
4. All packets are assumed to have a fixed size of 500 bytes, and all ACK packets a size of 50 bytes.

The simulation code of the TCP protocol is a simplified version of the TCP implementation from the BSD UNIX4.3 release (i.e. the connection set-up and tear-down parts are removed). ICMP source-quench (Postel, 1981b) is not activated in the simulation tests because there is no clear specification on how the source-quench message should be handled.

## 3. SIMULATION RESULTS: OSCILLATORY DATA TRAFFIC

This section presents the simulation results in graphical forms. We will focus on the results from Topology-1, because the topology is simpler and the results are easier to understand. The results from Topology-2 are used to confirm that a more complex system also exhibits oscillatory symptoms similar to those observed in the simple system.

### 3.1. Observations with Topology-1

Ten TCP connections are set up on this simple topology. Connections 1 to 5 send data from Host-1 to Host-2; connections 6 to 10 send in the other direction. The trace of the packet queue length at switches clearly shows an oscillating load (see Figure 2).

Looking into each connection's transmission behavior, we can see a corresponding oscillation of the congestion control window size of each connection. In particular, these

[2] Observations from simulations show that, with the Slow-Start congestion-control algorithm the exact pattern of packet generation is largely irrelevant as long as the average generation rate is not below the available channel bandwidth. Because newly generated packets are accumulated when the congestion window size is small, the resulting transmission pattern is similar to that of an infinite data source.

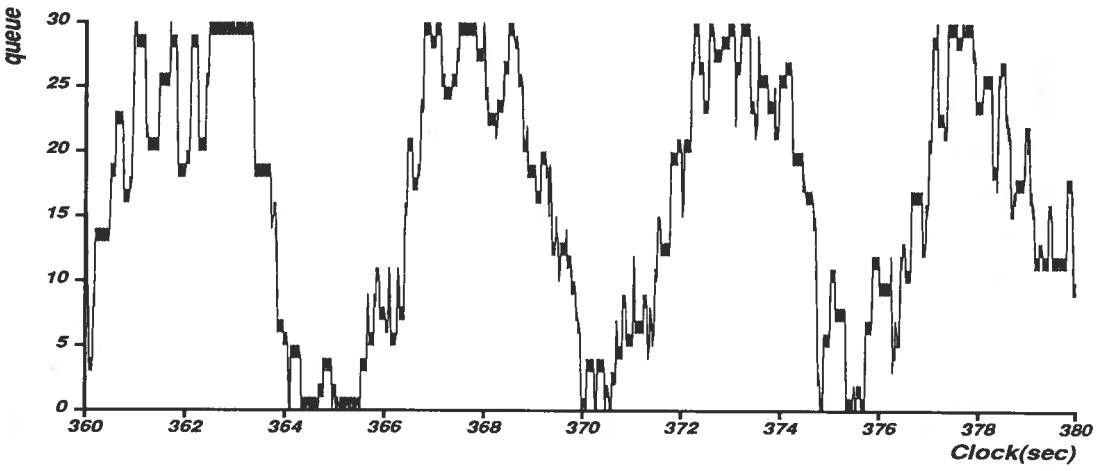


*Figure 2. A 20-second trace of packet queue length at Link-2, Switch-1*

window size oscillations are synchronized, as shown in Figure 3. It is this synchronized control window oscillation that results in the network traffic oscillation.

A trace of the end-to-end packet delays of connections 1, 3 and 5 shows oscillation as well (Figure 4), roughly corresponding to the packet queue oscillation at the switch. For example, the minimum end-to-end packet delay of connection 1 is approximately 50 ms. Owing to the oscillating packet queue at the switches, however, the average packet delay is 490 ms, and the delay deviation is 213 ms (computed from a simulation run over 10 minutes of network time).

## 3.2. Observations with Topology-2

Fifty TCP connections are set up in Topology-2. Among the fifty connections, connections 1 to 24 have a path of one-hop (i.e. across one inter-switch link only), connections 25 to 40 are two-hop, and the rest are three-hop connections. The sources and destinations of the connections are more or less uniformly distributed; and each of the three inter-switch links carries packet flows from all of the three path length groups. Despite the mix of traffic and a much larger packet buffer size (i.e. 100 per switch), data traffic oscillation similar to that observed in Topology-1 exists; however, the oscillation has a lower frequency (as shown in Figure 5).

# 4. ANALYSIS

Close observations of our simulations exposed that the traffic oscillation results primarily from a combined effect of three causes. First, every time a switch runs out of buffer space and starts dropping packets, it takes a long time to get out of the congested state. Much like an object of large mass in motion, packet traffic keeps coming at the original speed long after the switch entered the congested state, as if no packet dropping had occurred.

We call this phenomenon the *traffic inertia*, which can be attributed to the operation of the window flow control mechanism. Suppose that the $N$th packet of a connection, $C$,

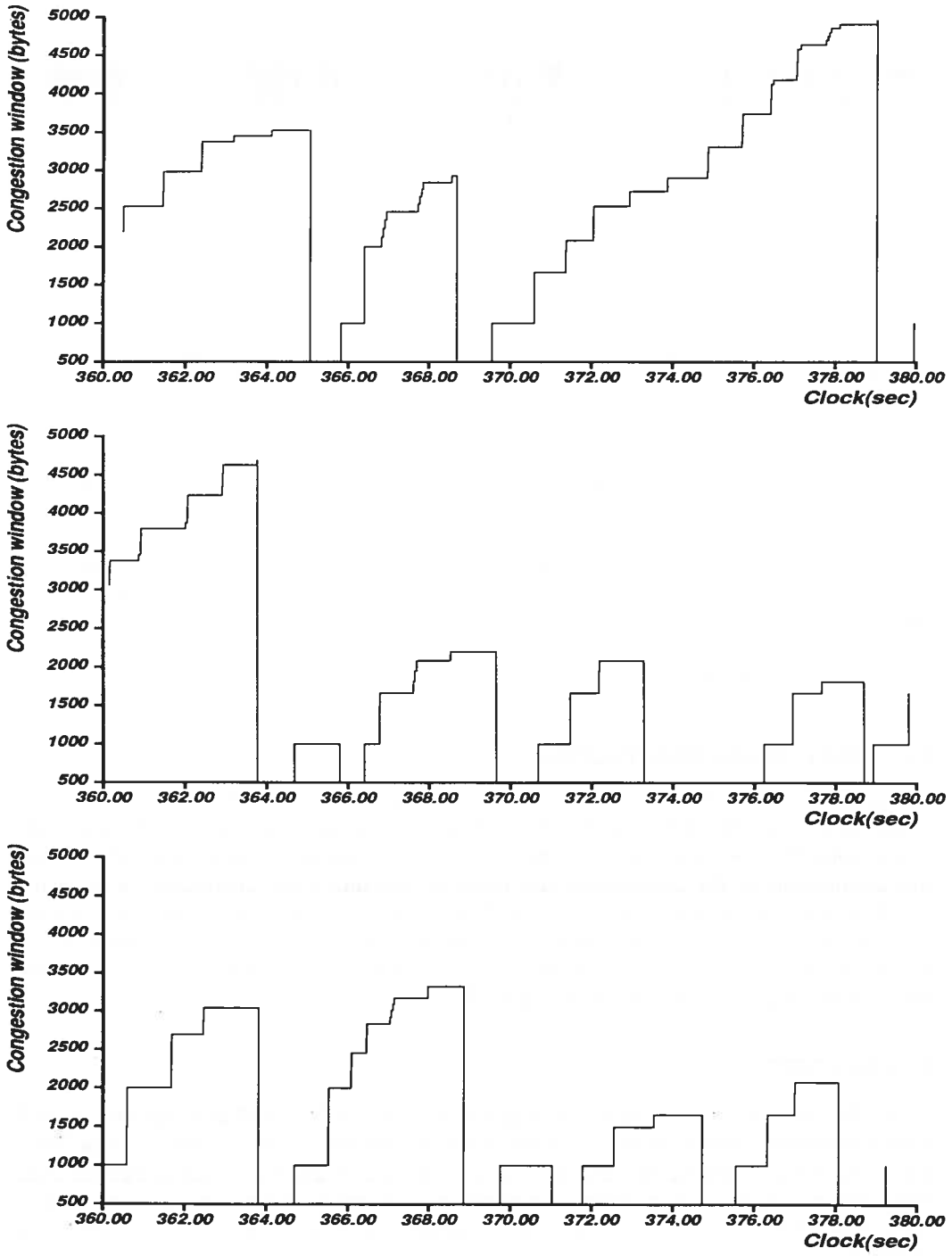


*Figure 3. A 20-second trace of the congestion control windows of connections 1, 3, and 5 in the Topology-1 test*

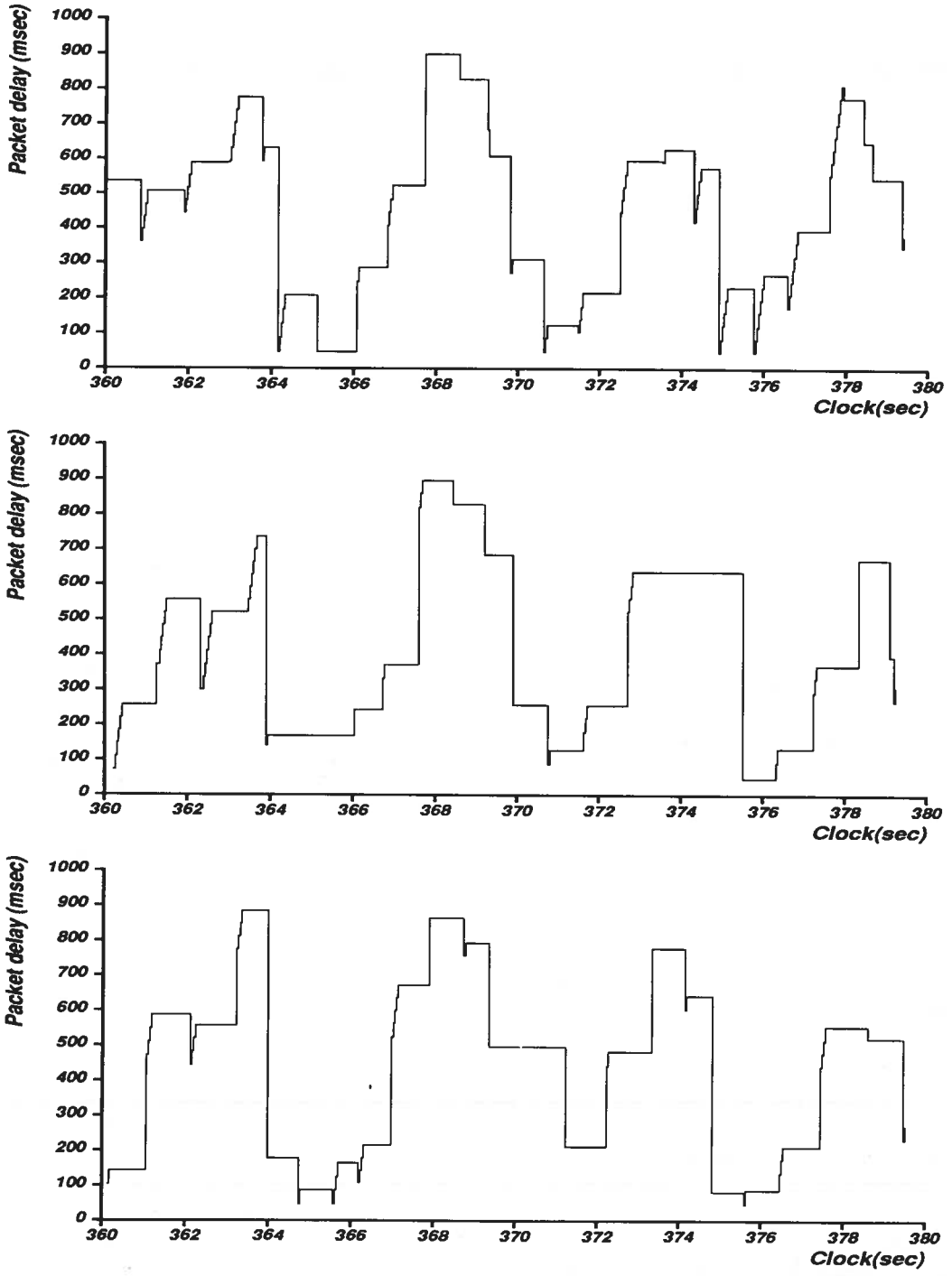


*Figure 4. A trace of one-way end-to-end packet delays of connections 1, 3 and 5 in the Topology-1 test*

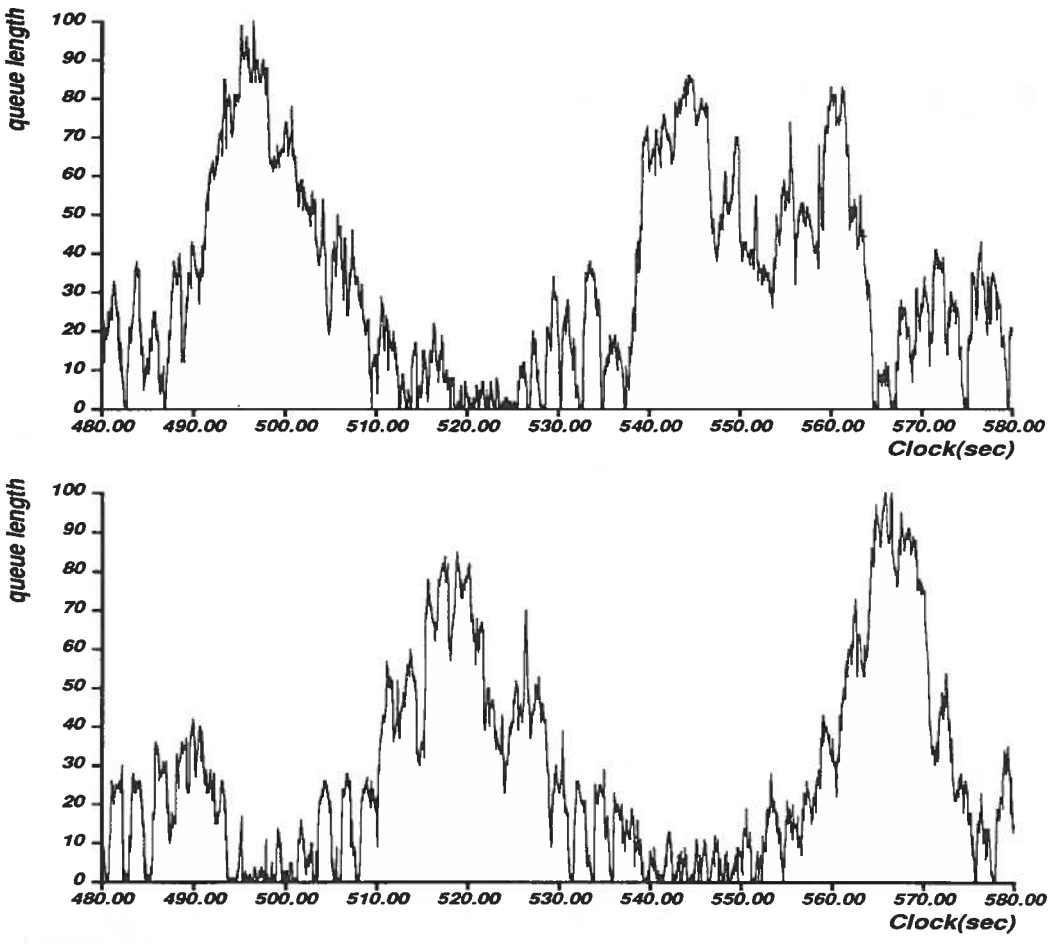


*Figure 5. A trace of the packet queues at Link-11, Switch-2 and at Link-13, Switch-3 in the Topology-2 test*

is lost, $C$ does not stop transmission until after the $(N + W - 1)$th packet is sent, where $W$ is the control window size (assuming that the other direction of the path is not totally blocked, so that the acknowledgments return successfully). Thus a single connection has a flow inertia equal to its window size, and the aggregate traffic has an inertia equal to the sum of the window sizes of all active connections.

This traffic inertia reflects the control delay in the Slow-Start algorithm. It determines the duration of time that the network is in a congested state. During this time period, most connections passing through the congested point lose packet(s), no matter whether a connection is directly responsible for the congestion (although some connections may lose more than others). For instance, from Figure 2 we can see that congestion occurred at Link-2, Switch-1 around the 363rd second of the simulation test (at this time the packet queue length has reached the buffer size limit of 30); Figure 3 shows that, around this time, connection 1 has a control window size about 3·5 Kbytes, connection 3's window

is around 3·5 Kbytes, whereas connection 5's is near 3 Kbytes. None the less all three connections lost some packets during the congestion event, as evidenced by the close-down of their congestion control window shortly thereafter.

Secondly, although individual connections are supposed to operate independently from each other, the simultaneous packet losses make the open/close cycles of their congestion control windows highly synchronized. Connections that lost packets at the same congestion point enter a retransmission time-out waiting state within a relatively short period. The network traffic empties out during this wait time period.

Thirdly, after the detection of congestion (as signaled by packet losses), it takes a rather long time to build up the network load again. Because the end-to-end connections receive no quantitative information other than packet losses as a congestion signal, the Slow-Start algorithm takes a conservative approach and reduces the congestion control window to one packet. A number of round-trips are needed to open up the control window to a proper size again. During this time period, the bottleneck links are under-used.

The above explains why a congested state of a switch is always followed by an empty state. Now let us see why the network enters a congested state repeatedly. This is due to the aggressive window opening strategy of users. All the TCP connections start with a minimal control window size of one packet. In the absence of packet losses, however, the window keeps opening up as a consquence of acknowledgment returns. The Slow-Start algorithm uses this 'toe-in-water' strategy to find out whether the highest possible throughput has been achieved. Because packet losses are the only congestion signal that can stop the speed-up, data traffic keeps increasing until it hits the network congestion point. After the congestion recovery, the same congestion control window opening cycle repeats, leading to the next congested state.

What we can conclude from the above analysis is that, owing to the traffic inertia phenomenon, the majority of users lose packets simultaneously at congestion, independent from whether they are responsible for the congestion, and that the synchronized packet losses in turn lead to synchronized control actions of all the users. Furthermore, because users have to acquire network capacity by continuously increasing their control window size, congestion occurs repeatedly.

We see that the ultimate cause of traffic oscillation is the aggressive speed-up followed by synchronized slow-down of connections upon packet losses. The speed-up in turn is due to the binary feedback information from the network, i.e. whether the network is congested or not. Therefore in the absence of packet losses users assume that further speed-up is always feasible, which leads to congestion. Owing to traffic inertia, packet losses caused by congestion hurts most users, forcing them to reduce the transmission rate to the minimum at the same time. Therefore, even when the number of active users remains the same, the network load fluctuates.

Such behavior can be corrected by providing users with selective feedback signals and with more quantitative network load information. The information provided by packet loss does not reflect reality. In reality only some of the users are major contributors to each congestion, and the network load is a continuous variable rather than a binary state. If these two kinds of information can be propagated to end users, they would be able to avoid both the synchronized control actions as well as excessive speedup once the network load reaches a proper level.

## 5. RELATED WORK: USING HEAVY DAMPING TO PREVENT OSCILLATION

Before concluding we relate our results to some previous work on this subject, mainly the DECbit congestion avoidance algorithm and a delay-based algorithm proposed by Jain (1989).

### 5.1. DECbit algorithm

Another well-known congestion control algorithm for datagram networks, DECbit (Ramakrishnan and Jain, 1988), uses a binary feedback scheme similar to Jacobson's. Switches in the network detect congestion and set a congestion indication bit on packets flowing in the forward direction. The congestion indication is communicated back to the senders through end-to-end acknowledgments. If at least 50 per cent of the congestion bits are set, the flow control window size is reduced; otherwise it is increased. Because the network state is considered to be a binary variable, users can only adjust their flow control window by trial-and-error, similar to TCP connections.

Simulation of DECbit, however, shows only minor oscillations in the vicinity around an ideal operation point.[3] This is because heavy damping is built into the control algorithm—when a heavy load is detected, the control window of each connection is reduced only by a factor of 0·875 per control cycle. A control cycle is about two round-trip times (RTT). The DECbit scheme takes control delay into account. After each flow control window adjustment, the connection waits for an RTT before taking another measurement which takes roughly another RTT time period.

Heavy damping, however, may not be a good solution to traffic oscillation. Because of the small adjustment step used, heavy damping may take too long to reach a desired control window size when network load changes rapidly, such as when active users terminate or new users start. For instance, simulation results presented by Ramakrishnan, Jain and Chiu (1988) show that, with a simple topology (four switches in a row), when a second user starts transmission, the first user takes more than 10 round-trip-times (RTTs) to adjust its window size to half.[4] This can be too long a control adjustment period in a high-speed network environment, not to mention the possibility of buffer overflow during this long period. In a gigabit network, many data transfer applications may have completed within a time period of a few RTTs.

### 5.2. Delay-based algorithm

Jain (1989) proposed a delay-based congestion avoidance algorithm which is similar to DECbit except that it uses a different congestion signal. Instead of using explicit congestion bits set by the network, Jain's algorithm uses implicit information derived from the RTT

[3] However, because only the trace of the connection's window size changes are presented and no measurement of the network traffic is mentioned in Ramakrishnan and Jain (1988), we do not know exactly how the switch queueing changes with time.

[4] This is a rough estimate based on the graph (Ramakrishnan, Jain and Chiu, 1988, Figure 8-b). Each control cycle takes about two RTTs, and the coefficient used for window adjustment is 0·875, $(0{\cdot}875)^5 = 0{\cdot}513$. Thus, a five-step adjustment needs a period of 10 RTTs.

measurement to determine whether a connection should increase or decrease its flow control window. To minimize the amplitude of oscillation, it uses the same increase and decrease parameters as DECbit. In addition to the slow convergence property described above, this algorithm may also suffer from the drawbacks associated with delay-based algorithms, as pointed out by Robinson, Friedman and Steenstrup (1990).

We suggest that one possible way to reduce the heavy damping without sacrificing stability is to derive a more quantitative control value from the available information. For example, depending on the percentage of the congestion bits set, the DECbit algorithm may compute a finer granularity value of the network state, avoid adjusting the flow control window size up or down all the time. The same approach can apply to Jain's delay-based algorithm.

## 6. SUMMARY

In a heterogeneous internetworking environment, severe bottleneck points are normal cases rather than exceptions. Therefore effective congestion avoidance and control algorithms are needed to prevent the network from congestion collapse (Nagle, 1984). It seems difficult, however, to implement effective control while maintaining traffic stability in a large-scale, distributed environment.

By using simulation tools we examined in detail the causes of traffic oscillation in a simple network setting. Our analysis shows that, first, users' control actions are highly synchronized by the network congestion signaling in use (i.e. packet losses); and secondly, providing users with a binary network state is not adequate and can lead to fluctuating traffic. Although our analysis is based on the specific congestion control algorithm used in TCP/IP networks, we believe that this oscillatory traffic phenomenon applies in general to the design of congestion control algorithms. That is, a selective feedback signal, instead of a synchronized signal, and quantitative network load information, instead of a simple congestion signal, should be provided to end users to help properly adjust individuals' transmission rate.

This is our preliminary step in studying network control dynamics. More research issues have been identified for future study. First, what kind of quantitative information should the control algorithm provide? Should it be based on the aggregate traffic or on the contribution of individual users?[5] Secondly, who should measure and compute this quantitative information? Should it be carried out at network switches, at the user end, or jointly by both sides? What will be the control delay, and what will be its effect in this context? Moreover, will different service disciplines at switches help stabilize data traffic? Preliminary simulation results have shown that a fair-queueing service at switches may help reduce the synchronization among TCP connections, because only packets from the connections with the largest control window size get dropped (Shenker and Zhang, work in progress). Our investigation into these issues will be reported in future.

Although we made specific suggestions to improve network stability, it is yet to be determined to what extent the observed oscillation is due to the specific congestion control algorithm being used, and to what extent oscillation is an intrinsic phenomenon in large-scale, distributed systems, and is thus unavoidable.

[5] Scott Shenker of XEROX PARC has also looked into this question (Shenker, 1990).

## References

Hashem, Eman (1989) 'Analysis of random drop for gateway congestion control', *M.S. thesis*, Massachusetts Institute of Technology, September.

Jacobson, V. (1987) Presentation at the IETF meeting, *Proceedings of the Sixth Internet Engineering Task Force*, Phill Gross (Editor), Boston Mass, April.

Jacobson, V. (1988) 'Congestion avoidance and control', *Proceedings of Symposium on Communication Architectures and Protocols*, ACM SIGCOMM, August.

Jain, R. (1989) 'A delay based approach for congestion avoidance in interconnected heterogeneous computer networks', *ACM Computer Communication Review*, **19**, (5), 56–71.

Nagle, J. (1984) 'Congestion control in TCP/IP internetworks', *ACM Computer Communications Review*, **14**, (4).

Postel, J. (1981a) *DoD Standard Internet Protocol*, Network Information Center RFC-791, SRI International, September.

Postel, J. (1981b) *DoD Standard Internet Control Message Protocol*, Network Information Center RFC-792, SRI International, September.

Postel, J. (1981c) *DoD Standard Transmission Control Protocol*, Network Information Center RFC-793, SRI International, September.

Ramakrishnan, K., Jain, R. and Chiu, D. (1988) 'Congestion avoidance in computer networks with a connectionless network layer', *Innovations in Internetworking*, Artech House.

Ramakrishnan, K. and Jain, R. (1988) 'A binary feedback scheme for congestion avoidance in computer networks with a connectionless network layer', *Proceedings of SIGCOMM'88.*

Robinson, J., Friedman, D. and Steenstrup, M. (1990) 'Congestion control in BBN packet-switched networks', *ACM Computer Communication Review*, **20**, (1), 76–90.

Seo, K., Crowcroft, J., Spilling, P., Laws, J. and Leddy, J. (1988) 'Distributed testing and measurement across the Atlantic packet satellite network (Satnet)', *Proceedings of ACM SIGCOMM'88.*

Shenker, S. (1990) 'A theoretical analysis of feedback flow control', to appear in *ACM SIGCOMM '90*, September.

Zhang, L. (1989) 'A new architecture for packet switching network protocols', *Ph.D. thesis*, Massachusetts Institute of Technology, July.