11-26-1985

Network performance monitors

Daniel Sorrentino

Rochester Institute of Technology
School of Computer Science and Technology

NETWORK PERFORMANCE MONITORS

A Thesis submitted in partial fulfillment of
Master of Science in Computer Science Degree Program

Daniel G. Sorrentino

Approved by:

Margaret M. Reek
Professor Margaret M. Reek (Advisor)

John L. Ellis
Dr. John Ellis

Daryl Johnson
Daryl Johnson

Date: November 26, 1985

NETWORK PERFORMANCE MONITORS

Daniel G. Sorrentino

I grant permission to Wallace Memorial Library of RIT to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Daniel G. Sorrentino

Daniel G. Sorrentino

Table of Contents

Table of Contents

Table of Contents

List of Figures

List of Tables

NETWORK PERFORMANCE MONITORS

Daniel Gabriel Sorrentino

ABSTRACT

Over the last 10 years both industry and academia have conducted extensive research and development in the areas of local area network management and performance. This thesis investigates some different ways of measuring the performance of local area networks and studies hardware and software systems designed to watch over network performance and events. These systems are termed "network performance monitors."

As part of the thesis, a network performance monitor has been implemented to monitor TOOLSNET, an Ethernet based local area network, supervised by the Engineering Tools Technology group at Compugraphic Corporation. This network performance monitor is called TOOLSNET ANALYSIS PACKAGE, or TAP for short.

TAP runs Monday through Friday, from 8 a.m. to 5 p.m. to monitor TOOLSNET performance during working hours. A summary of the measurements from one of these

5 day sessions is reported in this paper. This summary includes the following tables: one showing a system by system contribution to TOOLSNET traffic for the session; another that shows the amount the network was utilized each day of the session; another that shows the total number of random errors experienced by TOOLSNET during the session; another that shows asymptotic throughput utilization, perceived utilization, relative load, and average response time for each day of the session; another that shows source-to-destination traffic for each day of the session.

performance. Furthermore, "watchdog", programs and devices (network performance monitors) have been developed to apply LAN metrics in monitoring the performance of LANS and to assist LAN managers in perceiving current behavior, in predicting effects of changes to the LAN environment, and in analyzing problems in LAN hardware, and protocols. This thesis is an investigation of LAN metrics and performance monitors, and includes the design and implementation of TOOLSNET ANALYSIS PACKAGE (TAP), a network performance monitor that I wrote for TOOLSNET, a LAN located at Compugraphic Corporation.

The outline for this thesis is as follows: Chapter 2 will discuss several LAN metrics, Chapter 3 will discuss several network performance monitors, Chapter 4 will discusses the environment and the results of a five day, work-week, session of TOOLSNET performance as reported by TAP. In Chapter 5, I share some personal learning experiences in the development of a network monitor on behalf of other graduate students who may desire to do one of their own. Chapter 6 discusses some recommendations for future developers of TAP. Appendix A contains an over-view of TAP's design. Appendix B contains the manual pages and sample output from TAP. Appendix C contains an annotated bibliography and reading list. Appendix D is a glossary of terms. Appendix E contains the actual code of TAP.

2 Lan Metrics

This chapter discusses some popular metrics for determining LAN performance and network behavior, such as channel utilization (or efficiency) , throughput efficiency, average response time, and perceived efficiency. These metrics are discussed here with an eye toward their inclusion in network monitoring systems, discussed in Chapter 4.

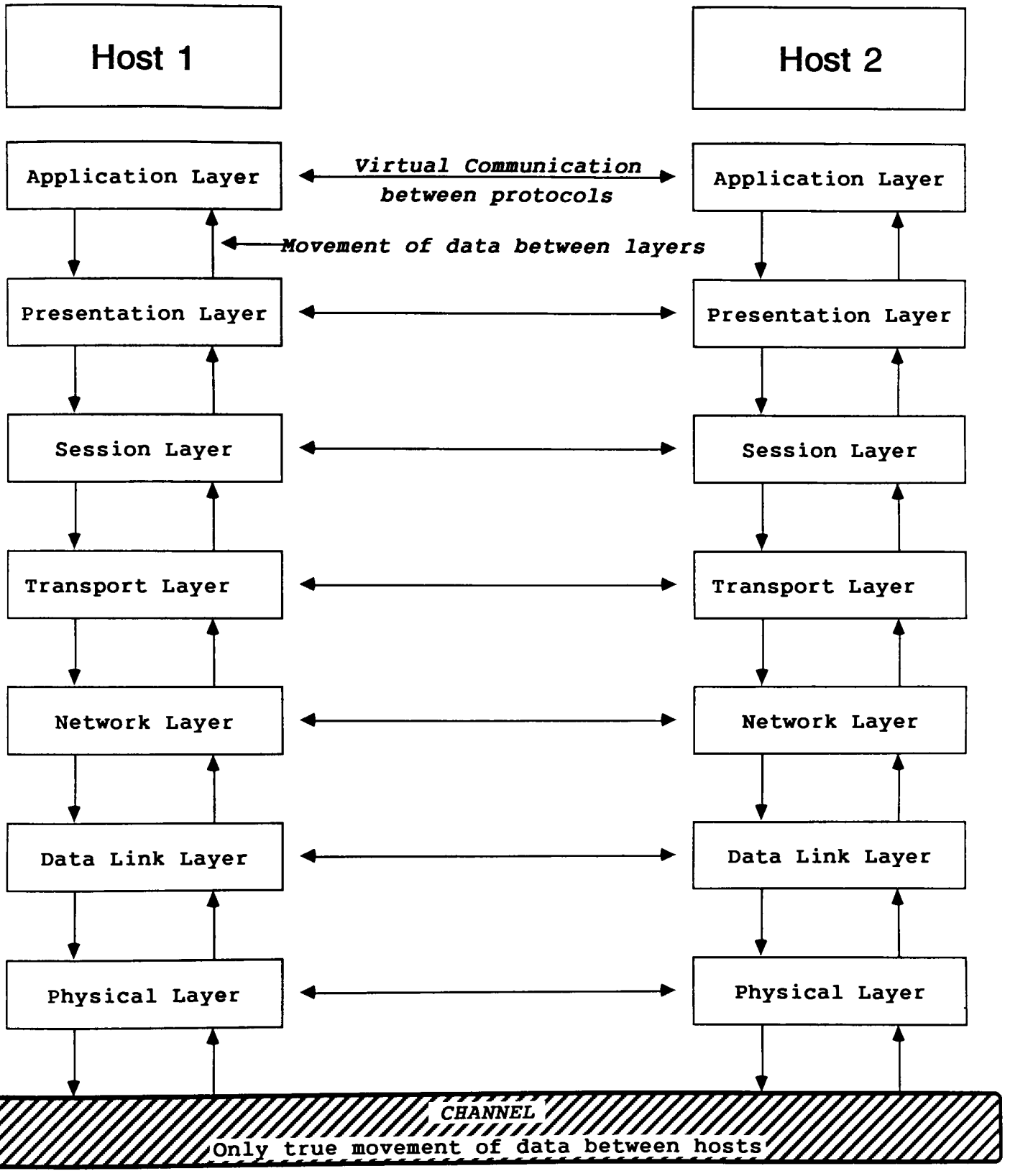
2.1 Delay

Local area networks are birth/death systems [TAN 81]. Packets are generated (born), transmitted, and extinguished after achieving their final goal. Hence, a packet has a lifetime. The shorter a packet's lifetime in the system, the faster its final goal, delivery of data, is achieved and the better the network is performing. This notion implies that the more delay that a packet incurs during its lifetime the poorer the network is performing. Delay has been a topic of study for networks of all kinds--the object often being to find out where the delays are happening and to reduce them where possible.

Queuing delays [KLE 64] [KLE 70] are caused by packets being forced to wait for a service by a limited number of servers. Examples of services that packets need include the acquisition of buffer space in which to accumulate or deposit data and the

building or removal of packet headers. Under the 7 Layered OSI model [SHA 83] (see Figure 1) there exists potential for a queueing delay at every layer. Packets may have to wait in 7 lines to have physical, data link, network, transport, session, presentation, and application layer services performed for them. Most LANS do not strictly follow the OSI approach; however, in most cases, there is some layering involved and a potential for queueing delays at each layer.

FIGURE 1
ISO-OSI Model of Network Layering



In order to measure queueing delays between layers, timing devices are needed that can time stamp packets, and the resolution of the devices must be in microseconds [MUR 84]. What is of interest here is the delay incurred when a packet arrives in a queue, waits for service, gets serviced, and is passed onto the next layer for service, repeating the same sequence at that new layer.

In their article [MUR 84], Murray and Enslow point out several specific kinds of queuing delay experienced in an ETHERNET, 10Mbps, Virtual Circuit LAN:

End-to-end character delivery delays are simply a measure of how long it takes data to be placed into a packet, transported over the cable, and passed onto the target port (Figure 2). In effect, it is a simple view of the birth and death of a packet. With network ports operating at baud rates of 1200 bps or 4800 bps they found that the average time required to do this peaked at 80ms or 61ms--more time than it takes to send a single asynchronous character over a leased line from Boston to San Fransisco! See Figure 3. Murray and Enslow broke down end-to-end delivery delays into the other delays which follow.

Figure 2.
(Adapted from [MUR 84])
Movement of packets through queues
and buffers endemic to virtual circuit protocols.

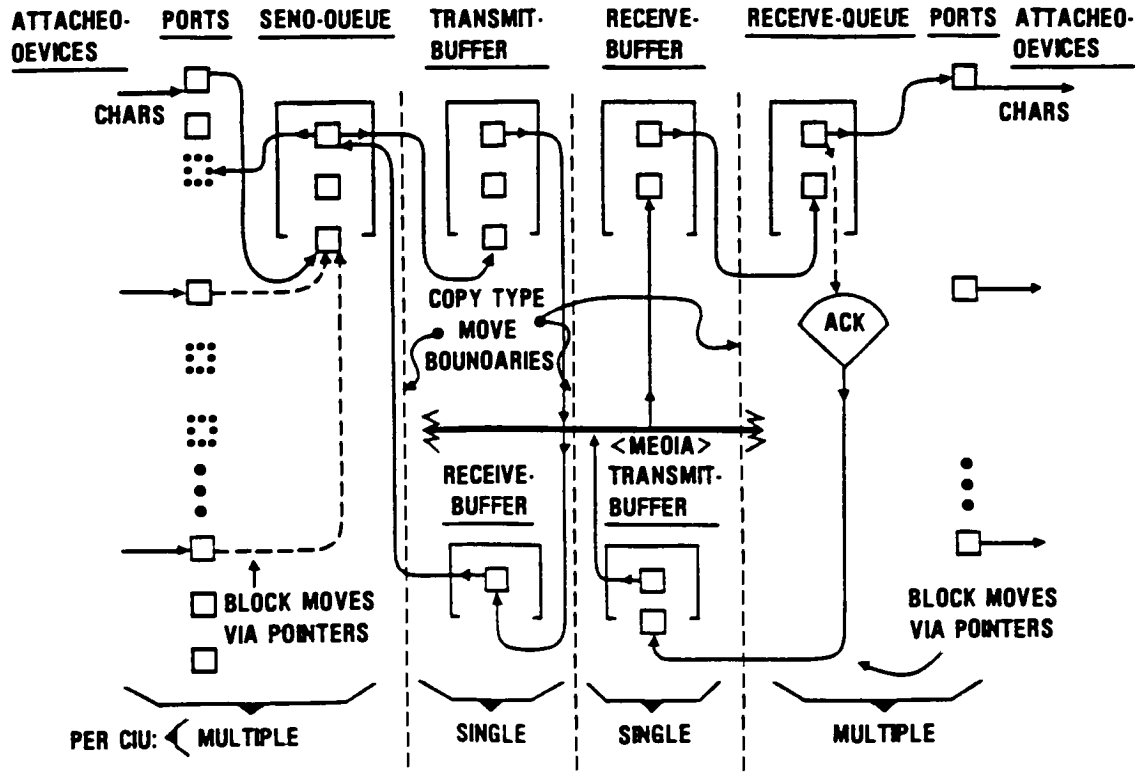


Fig. 2. The buffer management process.

Figure 3.
(Adapted from [MUR 84])
Character delivery delay distribution
showing delay peaks at 61ms and 81ms for
network ports running at 4800 and 1200 bps.

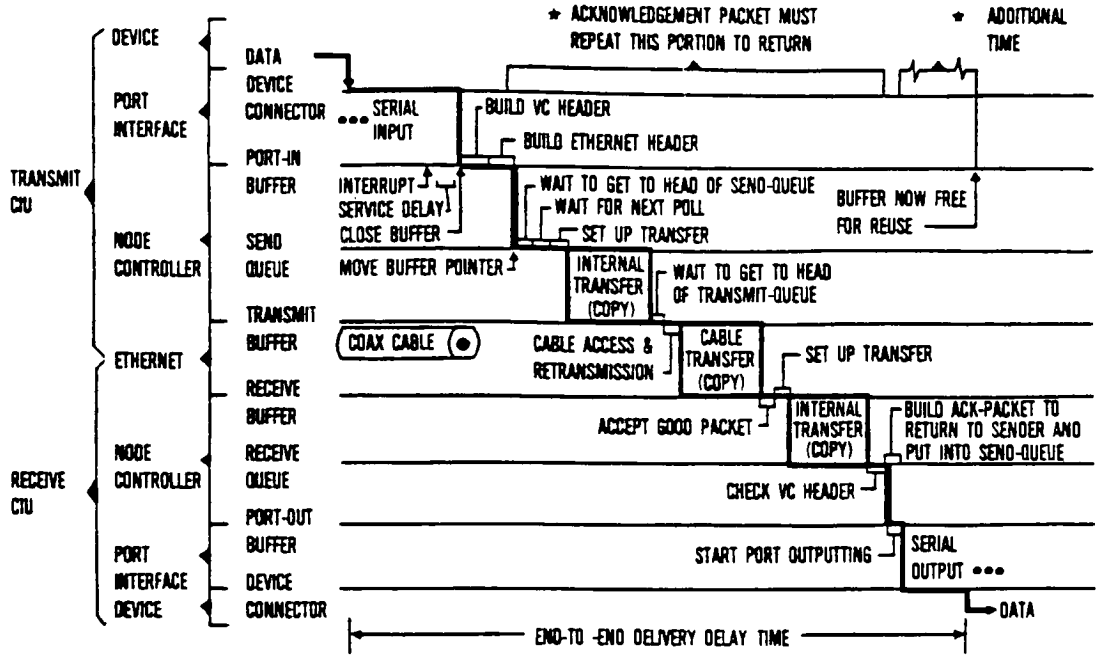


Fig. 3. Packet building and virtual circuit operations—Time Chart.

Packet building and buffer management delays endemic to virtual circuit protocols were regarded as the most important delays in the study. Performance suffered its greatest degradation in these two activities. Characters that need to be transmitted have to compete for a limited number of ports and buffers. A classic queueing delay [KLE 64] [KLE 70] is experienced at this point as queueing must wait for ports and buffers to become free for acquisition. Once a buffer is acquired for data another queueing delay is experienced as the buffer waits to move to the head of the send queue to be transmitted. As shown in Figure 4, 59% of the 80ms end-to-end delay from the 1200 bps ports occurred here.

Figure 4.
(Adapted from [MUR 84])
Break down of end to end
character delivery delays.

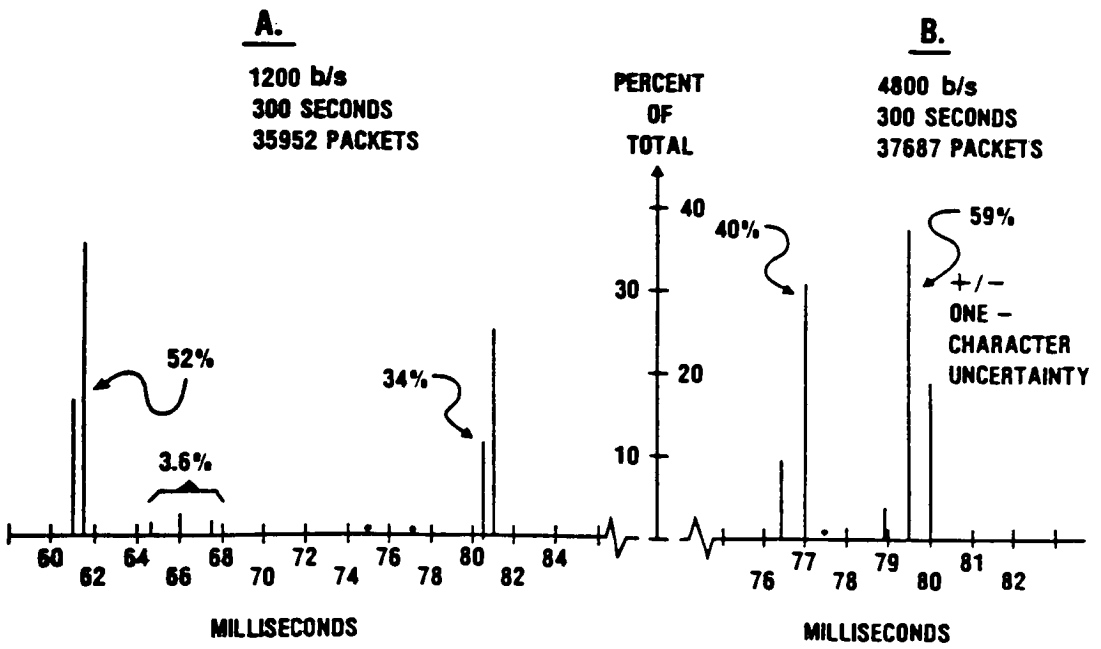


Fig. 5. Character delivery delay distribution.

Two serial transfer delays contributed significantly to the 80 ms total. An 8.3 ms delay was attributed to the movement of data between the device to the port buffer and another 8.3 ms delay was attributed to the time it took for the output transfer. These and other delays are also depicted in the end-to-end delivery delay chart, in Figure 3.

2.1.1 Comments

It should be noted that LANS are not usually used to package and send 1 character at a time. The preceding figures may be a little misleading if it causes us to look at a local area network as a character-by-character delivery system. The fact of the matter is that LANS deliver massive quantities of data more quickly than leased lines. Building a virtual circuit header may cost no more time for a packet of 1500 bytes than for a hypothetical packet of 1 byte. My own timing tests indicate that the larger the file being moved from one system to another, the faster the transfer rate. This indicates that the overhead involved in virtual circuit management is quickly compensated for by local area network technology. For example, transferring a 281600 byte file from one system to another over a 10Mbps, virtual circuit Ethernet took 9.3 seconds. This was a transfer rate of 33 microseconds per character, which is 1000 times faster than the rate reported for delivering a character over a leased line from Boston to San Fransisco. As another example, it took 1.3 seconds to

transfer a 1027 byte file. This was a transfer rate of 1.2 milliseconds to deliver 1 character. This is 30 times faster than the leased-line comparison above. In short, trying to compare the amount of data that can be transferred by a LAN verses a leased line in the same amount of time is like comparing the amount of water that moves down a river versus a garden hose.

2.2 Analysis of Ethernet-like Networks [ALM 79]

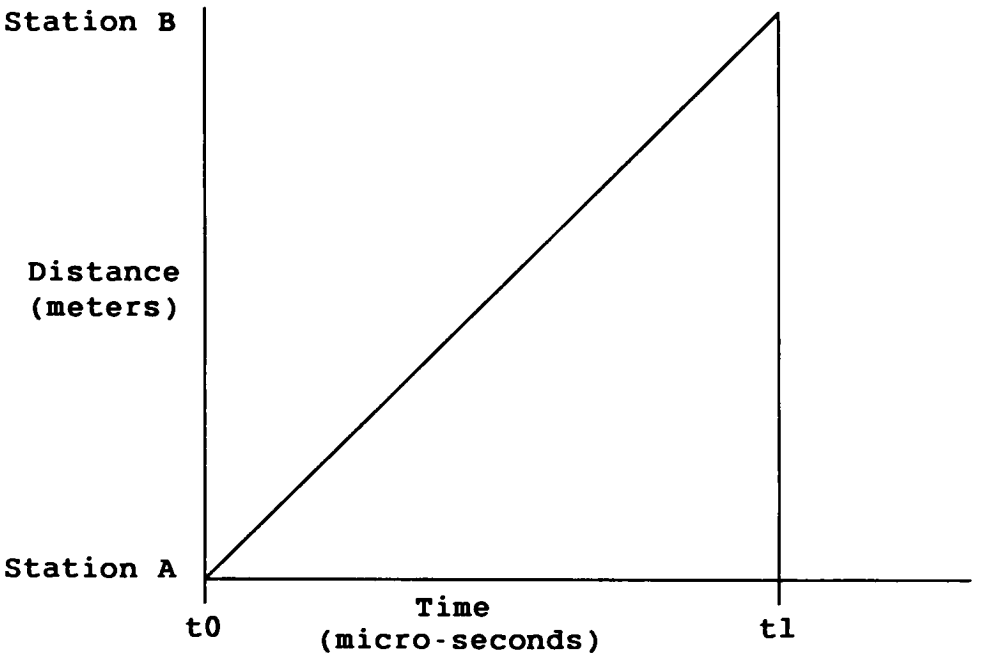
The discussion of delay could apply to Local Area Networks of varying topologies(see [CLA 76]). This chapter will discuss network metrics that specifically apply to Ethernet-like LANS, [MET 76], [ALM 79], [MUR 84], [SHO 80]. The network that is considered in this sub-chapter is akin to the original Ethernet explained in [MET 76]. That is, some number of computers have time division multiple access to a broadcast channel, the ether (for example, a coaxial cable) which is governed by a distributed control policy.

Channel time can be divided into three distinct categories: idle, that is no station is using the ether; transmission, that is one station has seized the ether and is sending a packet; contention, that is when more than one station is trying to transmit.

The distributed control policy is the means by which transmission is achieved and contention is resolved. If a sta-

tion is ready to transmit and senses an idle ether, it transmits. Disregarding random errors, (for example, checksum, frame errors, etc) in the time it takes the packet to traverse the channel one of two things can happen, either the packet will arrive safely its destination or it will suffer a collision. Collisions occur when one or more stations ready to transmit, sense an idle channel and transmit their packets simultaneously. Collisions also occur when one station (A) begins transmitting a packet while at least one other ready station (B) does not yet sense A's packet traveling on the channel due to signal propagation delay and transmits--thus resulting in a collision. Signal propagation delay is the amount of time it takes for a signal to travel from one point to another, and is directly proportional to the distance between them (see Figure 5). Once a collision is sensed by the stations involved, they abort transmitting their packets and enter into contention for the ether.

FIGURE 5



Time for signal to propagate between stations

If one station transmits at t0, and the other transmits anytime before but not equal to t1, there will be a collision.

Contention is resolved by the Binary Exponential Backoff Algorithm [MET 76] given below. Ethernet controllers divide retransmission intervals into slots which represent the maximum amount of time between starting a transmission and detecting a collision, a one end-to-end round trip delay. An Ethernet controller begins transmission of each new packet with a mean retransmission interval of one slot time. Each time a controller detects a collision it attempts retransmitting the packet after it has waited a random interval of time possessing a mean twice that of the previous interval, plus any time needed to wait for the channel to clear and become idle. This algorithm tries to insure that as a contending station (one of two or more stations involved in a collision) becomes ready to retransmit, the chance of doing so without collision is increased because the other contending stations are either in the waiting interval or have already attempted retransmission.

As stated above, in analyzing the performance of an Ethernet, time is divided into slots that are equal to the round-trip propagation delay. This is the amount of time that it takes for a station to be sure that its signal has not suffered a collision from the most distant station. The instantaneous load (Q) on the ether is considered the number of stations that desire to transmit during one slot. If no stations transmit (Q = 0), the slot is wasted and the ether is idle. If Q > 1 a collision occurs. If Q = 1, one station seizes the ether and transmits its packet, without collision.

In an Ethernet LAN each station has an equal chance at transmitting whenever the channel is free. There is no forced preference or priority among stations at any time. Metcalfe and Boggs [MET 76] describe the probability that one station acquires the ether in a given slot as:

$$A = (1 - 1/Q)^{Q-1} \qquad (1)$$

This means that A (acquisition probability) decreases, as Q (the number of stations ready to transmit) increases. As stated above, if Q is greater than 1 for a given slot there will be a collision and if it is less than 1, no station seizes the either. As Q increases, out to infinity, the acquisition probability approaches 1/e which is equal to 1/2.7 (. 37) or a 37% chance that the ether will be successfully seized.

The probability that the ether is acquired on exactly the i-th slot is

$$A(1-A)^{i}. \qquad (2)$$

Equation 2 can be explained by the following simple scenario. Suppose for an imaginary network of 2 computers that Q=2, (both stations are ready to transmit at the same time). Both stations transmit resulting in a collision. Both stations then wait a random amount of time before attempting to retransmit (during which Q=0). Due to this waiting period more slots go by wasted (i=2,3,4,etc.,). When one of the stations is done waiting

it transmits its packet without a collision (e.g. i=5), provided that the other station did not randomly wait the same amount of time, then the other station becomes ready and transmits its packet (e.g. i=9). This simple scenario demonstrates that if there are 2 or more stations ready to transmit, then more slots available to them (i increasing) results in a higher probability that they will acquire the channel. In other words, stations may not acquire the first few "free" slots that go by due to collision, but as more slots go by, their chances of acquisition increases with each slot. Of course, the scenarios become more complex as the number of hosts involved in the LAN increases.

The average number of slots devoted to contention (eq. 3) prior to acquiring the ether is expressed by the following equation:

$$Z = \sum_{i=1}^{\infty} iA(1-A)^{i} = (1-A)/A \qquad (3)$$

(Where SS = Sigma and oo = infinity.)

Z (the average number of slots) increases as A decreases due to more and more stations that desire to transmit (an increase in Q).

This equation leads into a popular LAN metric: channel utilization or channel efficiency.

2.3 Channel Utilization

Almes and Lazowska in [ALM 79] describe channel utilization as the instantaneous throughput efficiency of the network. That is, the ratio of the proportion of time the network is successfully carrying packets (transmission intervals) to the proportion of time the network is busy (transmission intervals plus contention intervals) when the instantaneous load Q is artificially held constant. Instantaneous throughput efficiency is expressed by the following equation:

$$U = \frac{(P/C)}{(P/C) + 2*tw*Z} \qquad (4)$$

where P is the average packet size in bits, C is the carrying capacity or data rate of the ether in bits per second, tw(greek: τow) is the propagation delay in seconds, multiplying it by 2 accounts for round-trip propagation delay, which is the slot time, and Z is the mean number of slots devoted to contention.

From equation 4 it can be seen that as Z increases with an increase in instantaneous load (Q) that the instantaneous throughput efficiency, U of the LAN decreases. Recall that Z increases as A decreases with a rise in Q. It can also be seen that as average packet size increases that U will increase. Tw can increase by merely lengthening the ether, which will decrease U. As C increases U will decrease since the capacity will exceed the ability of the LAN stations to utilize a significant portion of the bandwidth.

Table 1 displays instantaneous throughput efficiency for various instantaneous loads and average packet sizes, C = 3 Mbps, 2 x tw = 10 usec.

Table 1
(Adapted from [ALM 79])
Instantaneous throughput efficiency for
average packet sizes of 256, 512, and 2048
bits. Cable length = 1000m. Data rate = 3 Mbps.
Propagation delay = 200 m/sec.

	Bits		
Q	256	512	2048
1	1.000	1.000	1.000
2	0.895	0.945	0.986
3	0.872	0.932	0.982
10	0.844	0.915	0.917
50	0.835	0.910	0.976
100	0.833	0.909	0.976

The adaptive distributed control policy designed by Metcalfe and Boggs [MET 76] is inherently stable. This stability can be observed by calculating the asymptotic instantaneous throughput efficiency of the network. From equation (1) it was noted that as Q increases A approaches 1/e. If the packet size is held to the minimum size, that is one in which its transmission time equals the slot time, then:

$$U = \frac{1}{1 + \frac{1 - A}{A}} = \frac{1}{1 + (e - 1)} = \frac{1}{e} \qquad (5)$$

Put in simple terms, under heavy loads the throughput of the network will be at least 1/e times the network carrying capacity. When the packet size is increased then the asymptotic throughput efficiency should be much larger than 1/e.

[TAN 81] takes a slightly different approach to calculating instantaneous throughput efficiency (utilization), in which he considers an ethernet LAN in the contention state. Recall that when stations are in contention that the contention resolution algorithm assigns a random amount of time that a station must wait before attempting retransmission. Therefore, when a free slot comes up, there is a probability (p) that the station will transmit. The acquisition probability, A (eq. 1) becomes:

$$A = Qp(1 - p)^{Q-1} \qquad \text{eq. 6}$$

A is largest when p equals 1/Q; that is 1 station out of all the stations in contention actually gets its packet out onto the channel. The value for A-->1/e as Q-->oo. Just plug some big numbers for Q into eq. 6 to prove this. The probability that the contention interval has exactly j slots is:

$$A(1 - A)^{j-1} \qquad \text{eq. 7}$$

So the mean number of slots per contention under heavy and constant loads is:

$$Z = \sum_{j=0}^{oo} jA(1 - A)^{j-1} = \frac{1}{A} = 1/(1/e) = e \qquad \text{eq. 8}$$

(S = sigma and oo = infinity) Consequently, the previous equation for utilization (eq. 4), mentioned in [ALM 79] is converted to:

$$U = \frac{P/C}{P/C + 2*tw*e} \qquad \text{eq. 9}$$

To briefly summarize eq. 9, in dividing time into slots, each slot on Ethernet has a duration of 2(round trip) * tw, and that the mean number of contention slots available is never more than natural 'e', (roughly 2.7). Consequently the deterioration in utilization due to propagation delay and contention for Ethernet is never more than 2 x e x tw or 5.4tw, even under heavy and constant loads. He goes on to report that tw = 5 microseconds for a 1 km cable. With P = 1000 bits utilization of the cable will be:

$$U = \frac{1000/10\text{Mbps}}{1000/10\text{Mbps} - 5.4 \times 5 \times 10^{-6}} = 1\%$$

As you can imagine, LANS that have channels with high bandwidths rarely utilize anywhere near 100% of the capacity.

Stallings in [STA 84] claims that the two most useful parameters in analyzing local area networks are the data rate (C) of the medium and the average signal propagation delay (tw). He claims that the product of Cx(tw) is the most important factor in determining the performance of local area networks. Under identical conditions a 50-Mbs, 1-km bus will exhibit the same utilization as a 10-Mbps, 5-km bus. The article points out that the product of Cx(tw) is equal to the length of the transmission medium in bits, that is, the number of bits physically out on the ether and traveling between two nodes at a given instant. Hence both of the networks mentioned have the same bit length, 50 bits.

[STA 84] develops yet another equation for channel utilization in the following manner. The length of the medium, expressed in bits, compared to the average packet length is denoted by a:

$$a = \frac{\text{Length of Data Path in Bits}}{\text{Average Packet Length(P)}}$$

or

$$a = \frac{C \times tw / P}{P}$$

P/C is the time it takes to get the average packet out on to the network. So,

$$a = \frac{\text{Propagation Time}}{\text{Transmission Time}}$$

Some calculations of 'a' for Ethernet LANS of different lengths, data rates, and average packet sizes are given in Table 2.

Table 2
(Adapted from [STA 84])
Values of 'a' for different average packet sizes, data rates, and cable lengths.

Data Rate	Packet Size	Cable Length	a
1 Mbps	100 bits	1 km	0.05
1 Mbps	1000 bits	10 km	0.05
1 Mbps	100 bits	10 km	0.5
10 Mbps	100 bits	1 km	0.5
10 Mbps	1000 bits	1 km	0.05
10 Mbps	1000 bits	10 km	0.5
10 Mbps	10000 bits	1 km	0.05
50 Mbps	10000 bits	10 km	0.05
50 Mbps	100 bits	1 km	2.5

Utilization is defined as the throughput of the channel divided by the data rate.

$$U = \frac{\text{number of bits transmitted}}{\text{number of bits that could be transmitted}}$$

$$U = \frac{P/(\text{Propagation} + \text{Transmission Time})}{C}$$

$$U = \frac{P/(tw + P/C)}{C} = \frac{1}{1 - a}$$

When the number of slots devoted to contention and round trip propagation delay is factored into the calculation:

$$U = \frac{1}{1 + 2aZ} = \frac{1}{1 + (2*tw*Z)/(P/C)} \qquad \text{eq. 10}$$

So far three different equations for utilization have been presented:

$$U = P/C \;/\; (P/C + 2*tw*Z) \qquad \text{eq. 4,}$$

$$U = P/C \;/\; (P/C + 2*tw*e) \qquad \text{eq. 9,}$$

$$U = 1 \;/\; (1 + (2*tw*Z)/(P/C)\;) \qquad \text{eq. 10}$$

In fact, they are all different forms of calculating the same measurement and are equivalent. As stated in its presentation above, eq 9 is a worst case scenario where the LAN is under heavy loads in which Z = 'e'. The difference is that eq. 4 can vary U

with different values of Z whereas eq. 6 fixes Z at e. Infact either factor, 2*tw*Z or 2*tw*e, has such a negligible effect on the final outcome of the calculation for utilization that it really does not matter which one is used; with Ethernets running at 10Mbps, propagation delay is so small (e.g. 2 microseconds) that multiplying it by any number between 0 and e that represents the mean number of slots spent in contention, hardly affects the final value of U. Given the 10Mbps LAN exemplified in Chapter 2.3 the reader can prove this to himself if he so desires.

Equation 4 and 10 are actually different forms of the same equation. Equation 4 is equation 10 with P/C factored into it--if each member of eq. 10 is multiplied by P/C it is transformed into eq. 4. Both will result in the same value for U.

Contrary to the opinion in [STA 84], [TAN 81], and [SHO 80], Almes and Lazowska in [ALM 79], argue that utilization is not a very meaningful performance measurement because Q, the number of stations ready to transmit, must be held artificially constant. They prefer to base other, more meaningful metrics on the average load , Ro , experienced by the network.

Average load is defined in [TAN 81] as the ratio of packets entering the network to the number of packets leaving the network during a given time period. In other words, it is the ratio of arrival rate of packets into the network to the mean service rate of the network hosts. The mean service rate is equal to 1 divided by the mean packet length in bits. Given an average

packet length P, stations submit packets onto the ether at an average rate of Ro*C/P per second. Almes and Lawozowska use these values to determine the following metrics:

o the proportion of capacity used to resolve contention times,

o Throughput efficiency of the network. That is the ratio of the proportion of time the network is successfully carrying packets to the proportion of time the network is transmitting and contending.

o Average response time of the network. That is, the average amount of time it takes for a station to successfully transmit a packet, after the point at which it was ready to transmit it. Note: I think that this is a poor definition for this metric. Almes and Lazowska look at this metric as if the ether "responds" to a stations desire to send a packet and that the amount of time a station has to wait around is the measure of the response time of the ether. It seems that this metric should be called average WAITING time, instead. This will become even clearer in the following equations.

o Perceived efficiency of the network. That is, the ratio of the theoretical transmission time for a packet of length P to the average response time seen by a station transmitting packets of that size.

In [ALM 79] an Ethernet simulation, using a Markov model, was conducted to determine the proportion of time spent in

transmission, contention, and idle states for varying average loads for packets of 256 and 2048 bits, and for a carrying capacity of 3 Mbps. The results are shown in Figure 6. As the average number of packets submitted onto the network over time increases, the number of transmissions increases proportionally until the asymptotic throughput efficiency (Eq. 5) is reached for that packet size. Beyond this point, the network enters the contention state. In other words, Figure 6 shows that as packets are infrequently submitted onto the channel (i.e. small average load per small proportion of time) that few collisions occur and that most transmissions succeed. As the number of transmissions increases, so does the offered load and the number of collisions, thereby resulting in more stations spending time in contention resolution. Notice that the graph with the smaller packet size spends more time in contention resolution. This is due to the fact that with shorter packets there is a shorter transmission time and more slots available for collisions.

Figure 6.
(Adapted from [ALM 79])
Ethernet time divided into
idle, contention and transmission time.

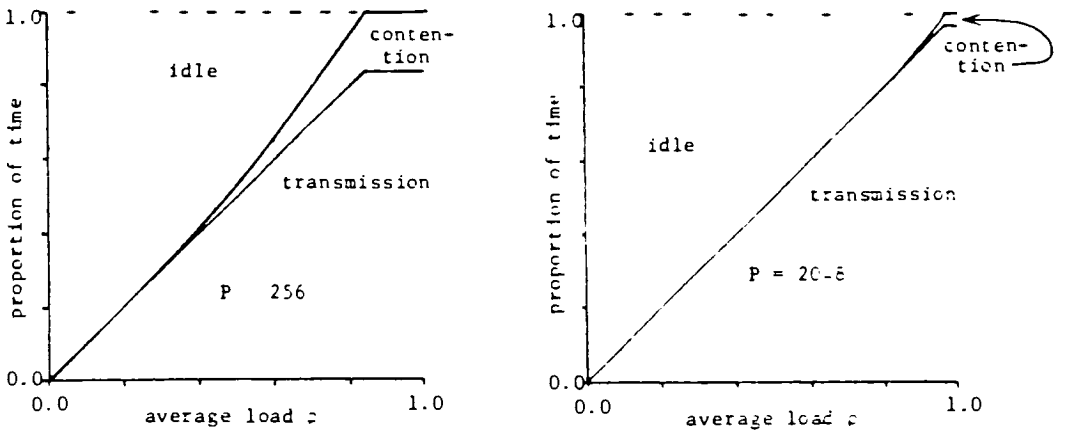


Figure 2-1: Time Transmitting, Contending, Idle

The simulation also produced results for average response times, as shown in Figure 7. As average load increases the response time slowly increases, until the average load reaches the asymptotic throughput efficiency. Then a sharp increase in response time is manifested, as shown by the knee in the curve. Here the larger packet size is more detrimental to this performance measurement.

Figure 7.
(Adapted from [ALM 79])
Average load vs. average response time.

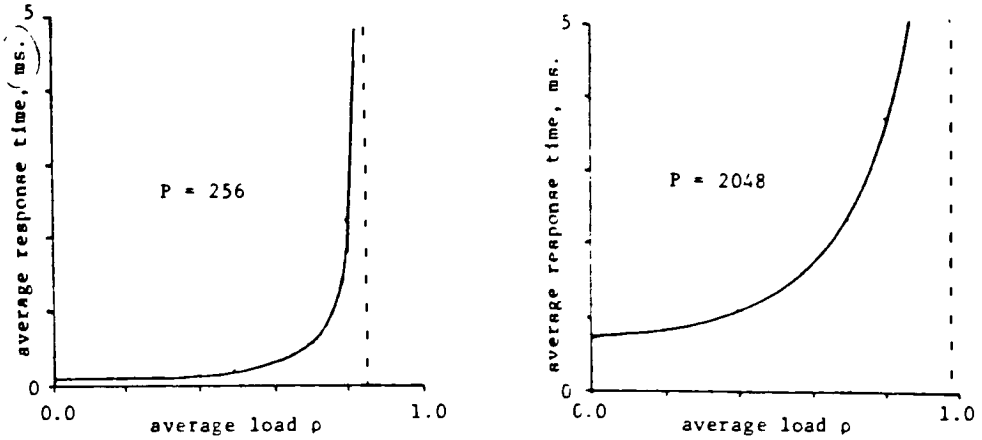


Figure 2-2: Average Response Times

The simulation also produced results for perceived efficiency; that is, the ratio of theoretical packet transmission time to average response time for packets of average length P. As shown in Figure 8, perceived efficiency decreases linearly with increased average load and reaches zero for an average load equal to the asymptotic throughput efficiency. At this point, stations are attempting to send packets faster than the network's ability to carry them and most of the time is spent in contention resolution.

Figure 8.
(Adapted from [ALM 79])
Perceived efficiency vs. average load.

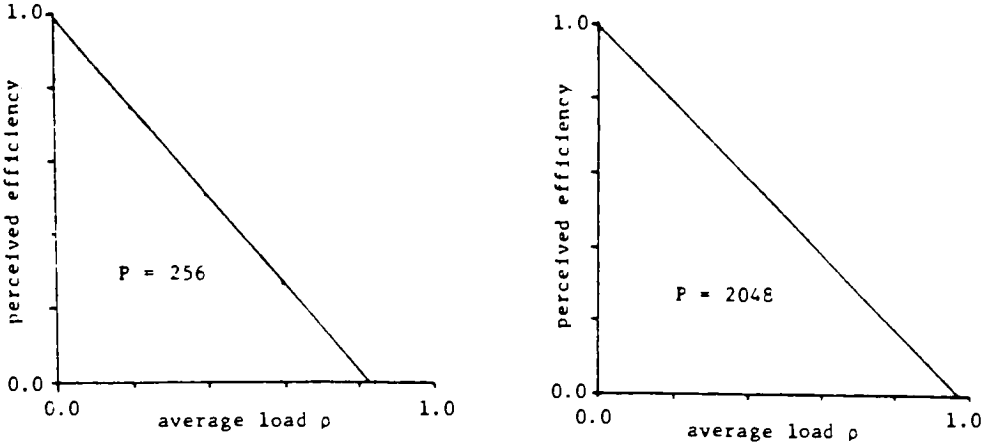


Figure 2-4: Perceived Efficiency vs. Average Load

Relative load is defined as the ratio of the average load, Ro, to the asymptotic throughput efficiency, for a packet of average size, P.

Almes and Lazowska observed that for a specific relative load, that perceived efficiency is equal to 1 - (relative load). This makes perceived efficiency much easier to measure and calculate. Other calculations become simpler through this observation. For example, given a network with an observed average packet size of P, an observed average load of Ro, and and asymptotic throughput efficiency of U, the relative load is

RL = Ro / U (11)

and the perceived efficiency, is

PE = 1 - RL. (12)

Linking this new definition of perceived efficiency (1 - relative load) with the old one which stated that perceived efficiency is the ratio of theoretical packet transmission time (average packet size / carrying capacity) to average response time for a network with a packet of average size P, the average response time for packets of size P is

AR = P / C / PE. (13)

In their example, using an average packet size of 512 bits, an average load of 0.4, and a calculated asymptotic throughput efficiency of .91

RL = .44 = 0.4 / .91,

PE = .56 = 1 - 0.44, and

AR = 305usec = 512 / 3Mbps / 0.56

2.4 Comments

The final goal of Almes and Lazowska's measurements is the calculation of average response time, which should be called average waiting time. Although this measurement is useful, it really only measures how long stations had to wait for the channel before transmitting. The importance of this measurement has been greatly diminished by today's technology. Most ethernet's run at 10 Mbps as opposed to 3 Mbps. Therefore the amount of time a station must wait for the channel to clear before transmitting has been drastically reduced; packets are on the channel for a short time. Although average response time is a delay statistic, the other delay statistics mentioned in the sub-chapter 2.2 give a truer sense of performance.

2.5 Summary

We have briefly discussed two major areas of LAN performance metrics: delay and utilization. Most of the delay incurred by LANS is found in packet building and buffer management services. Two approaches to calculating network utilization were discussed:

[TAN 81] vs. [STA 84] and [ALM 79]. The difference between these two approaches is that the former (in eq. 9) fixes the mean number of slots devoted to contention to natural 'e', whereas the latter (in eqs. 4 and 10) vary the number of contentions slots with the number of stations the desire to transmit. This difference is insignificant for two reasons: the mean number of slots is never more than natural 'e' and the propagation delay is so small that it practically diminishes the impact of contention times on utilization to zero.

Almes and Lazowska [ALM 79] argued that calculating channel utilization was not very useful because the number of stations ready to transmit (Q) must be held constant. In order to explore more meaningful metrics, they developed a Markov simulation that determined the proportion of capacity used to resolve contention, throughput efficiency, average response time and perceived efficiency of an Ethernet. A key observation was noted: Perceived efficiency is equal to 1 - Relative load. From this relationship, average response time became a simple calculation.

3 Network performance monitors

The last chapter discussed metrics for measuring the performance of local area networks. This chapter discusses LAN performance measurement at a higher level, network performance monitors. Network performance monitors are systems consisting of both hardware and software designed to watch over the behavior and performance of a LAN. Behavior characteristics include aspects such as the monitoring of source-destination traffic, whereas the calculation of delay or utilization would be a performance characteristic.

Most of the literature found in academic journals deals with LAN performance metrics, as discussed in Chapter 3 and in queueing theory and simulations [CHL 84], [HEY 83], and [NUF 83]. While network performance monitors do incorporate these metrics, merely studying metrics does not shed light upon aspects of design strategy and the variety of implementations for network monitoring systems, the final goal of this thesis.

Nevertheless, journal publications are not completely void of discussing network monitors. [AME 82] reports on a "measurement center" for a LAN used at the National Bureau of Standards; [MAT 82] reports on a network administration and control system for a broadband local area communications network; and [SHO 80] reports a fairly detailed study of a specific Ethernet installation by using a network monitor as an experimental base.

Proprietary literature dealing with network monitoring is another valuable source of information. Operating systems, such as UNIX 4.2 BSD come equipped with network monitoring software. Devices such as the EXCELAN Nutcracker [EXC 83] have been developed to monitor and experiment with Ethernet LANS. Software packages, such as the NETMAN utility by INTERLAN [INT 83], have been written to work with INTERLAN Ethernet controller boards, thereby making them more attractive to purchase.

3.1 A LAN MEASUREMENT CENTER [AME 82]

This article was so thorough in background information, concerning network monitor design, that it is appropriate to include much of its information in the early discussion of network performance monitors. This article discusses a network monitor used a the National Bureau of Standards. The LAN MEASUREMENT CENTER is a comprehensive and excellent model for developers of LAN monitors.

A LAN measurement center should have three components: a monitoring system, data analysis software, and an artificial traffic generator. The monitoring system should collect measurement parameters (e.g. size of packets, number of packets received, etc,.). The analysis software summarizes the information collected by the monitoring system and publishes performance reports which include information such as network delays, traffic distributions, and types of traffic transmitted. An artificial

traffic generator places varied loads on the network thereby allowing controlled experimentation.

Three design strategies were considered for the measurement center (i.e network monitor) for NBSNET: centralized measurement, decentralized measurement, and hybrid measurement.

A broadcast network conveniently lends itself to centralized measurement. A network controller could be forced to run in promiscuous mode in which the network controller examines every packet broadcasted on the network and monitors network performance from one location. Information such as interarrival times, packet sizes, and the addresses of source and destination hosts are available. Although a centralized monitor is cheaper than the other two approaches and introduces no traffic onto the network it has some drawbacks. It cannot determine how many or which stations are involved in collisions, nor can it report the arrival of packets onto the network without bias in its time measurements due to propagation delay of the signal from the sending hosts. Another problem occurs when traffic is so great on the network that a controller running in promiscuous mode cannot keep up with it, resulting in the loss of measurements.

In a decentralized measurement scheme each host contains a monitoring system and has real time clocks at each network interface to time stamp departing packets and record the time of incoming packets. This would overcome the bias problem mentioned in the centralized scheme. Furthermore, the monitoring system at each node could record both when the packets it transmitted

collided and the delay experienced by the node prior to retransmitting the packet.

Periodically, each node transmits its raw statistics to a central collector that processes them. This transmission is passed either over the network, generating traffic that will be monitored, or over separate channels, such as 9600 baud twisted pair wires. Either of these options has a disadvantage, the former in the generation of traffic on the LAN, the latter in additional cost. Furthermore, there may exist the classical problem of synchronization of timing devices at each node. This problem will surface when experimental requirements exceed the tolerance (or resolution) of available timing devices. Another disadvantage to a distributed measurement system is that each node has to supply memory and processor power to support it, either of which may be a scarce resource. Also, maintenance is more expensive than with centralized systems. Finally, since the entire system is distributed any change made to it has to be made at each node. This can be a tedious task, especially as the number of nodes increases.

Because of the disadvantages and advantages of centralized and decentralized network performance monitors a hybrid measurement approach was chosen for monitoring NBSNET. A hybrid approach tries to exploit the advantages and minimize the disadvantages of the earlier two alternatives. A hybrid approach requires that as much information as possible be measured in promiscuous mode at a centrally located node. This allows a reduction in the amount of

information that has to be gleaned at the "non-central" nodes, thereby decreasing their memory, processor, and traffic costs, while allowing all the advantages of the decentralized system.

The major disadvantage of a hybrid approach is the coordination of centralized and decentralized measurements, yet this can be overcome in part with careful design. Another disadvantage is that the centralized network controller may not be able to keep up with long bursts of traffic. This can significantly affect algorithms which analyze data. The bias problem of centralized measurement is present, but tollerable because the propagation delay is less than 10 microseconds for the most distant station. This is tolerable for the needs of NBSNET.

3.1.0.1 Measurement Reports

The hybrid Measurement Center generates ten performance reports for describing NBSNET traffic. A subset of them are described below:

Host Communication Matrix

This is a table showing the number of packets, data packets, and data bytes transmitted from each source-destination pair. Information such as this can help system administrators to relocate users on different hosts, distribute resources, pinpoint abusive users, and locate and kill use-

less servers.

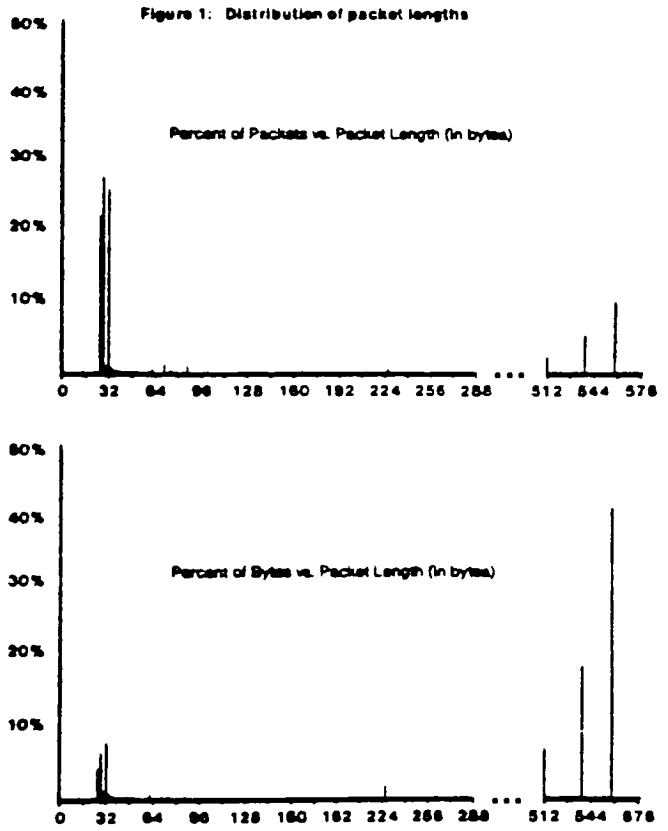
Data Packet Size Histogram

This records the number and proportion of data packets of particular length classes (ranges). For example, it might report X packets falling in the range of 100-200 bytes, Y packets falling in the range of 1000-1200 bytes, etc. In Figure 9, two histograms are included. The first histogram (Percent of Packets vs. Packet Length) displays the frequency of occurrence of various packet sizes on the LAN. This exemplifies the frequent arrival of acknowledgement and terminal traffic. The second histogram (Percent of Bytes vs. Packet Length) displays the traffic load characteristics. Shoch and Hupp in [SHO 80] concluded that most traffic on the LAN was comprised of file transfers, due to the large volume of traffic generated by the larger (512 bytes or greater) files. Hence, packet length histograms can indicate for what purpose the LAN is being used.

Figure 9.
(Adapted from [SHO 80])

- First Histogram: percent of packets vs. packet length (in bytes), shows frequency of packet size on the network. This indicates frequent traffic from terminals and acknowledgement packets.

- Second Histogram: percent of bytes vs. packet length, shows what sizes of packets contribute the most traffic on the net.



Throughput-Utilization Distribution

This distribution indicates the beneficial usage of the channel. A summary report includes the total channel throughput, channel utilization, information throughput, information utilization, and the number of seconds in the measurement period.

Packet Interarrival Time Histogram

Interarrival times are monitored centrally for each type of packet. An interarrival time is an important measurement for network metrics that use processes theory. It also stands on its own as a gauge for performance.

Channel Acquisition Delay Histogram

This is a report of the amount of time spent by LAN controllers for and acquiring the channel.

Collision Count Histogram

This histogram tabulates the number of collisions a packet encounters prior to a successful transmission. Reported as a summary, are the total number of packets transmitted, collisions and the mean number of collisions per transmission.

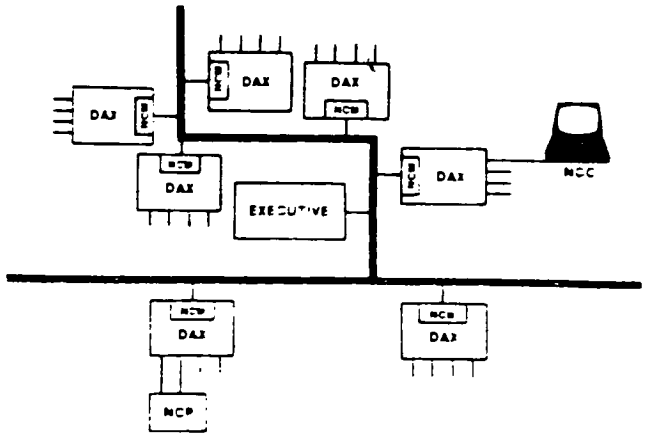
3.1.0.2 Comments

As you can see, the Measurement Center for NBSNET is a comprehensive network management system, possessing sophisticated timing hardware, an artificial traffic generator, and effective statistics gathering and processing strategies--combined, they give network administrators and managers a powerful tool for monitoring, maintaining, experimenting and improving the networking environment.

3.1.1 Network and Administration Control System (NAC)

The AMDAX CableNet(tm) Switched Service Local Area Communications Network [MAT 82] is a broadband based system that can support up to 16,000 end-user devices, with a maximum of 50 miles between the two most distant ones. A standard CATV coaxial cable is used. The bandwidth is 14Mbps CableNet employs Executive(tm) and DAX (Data Exchange) intelligent network controllers as network nodes. NAC is comprised of several components described below. An overview of the hardware components of NAC is displayed in Figure 10.

Figure 10.
(Adapted from [MER 82])
Overview of CableNet showing major hardware components.

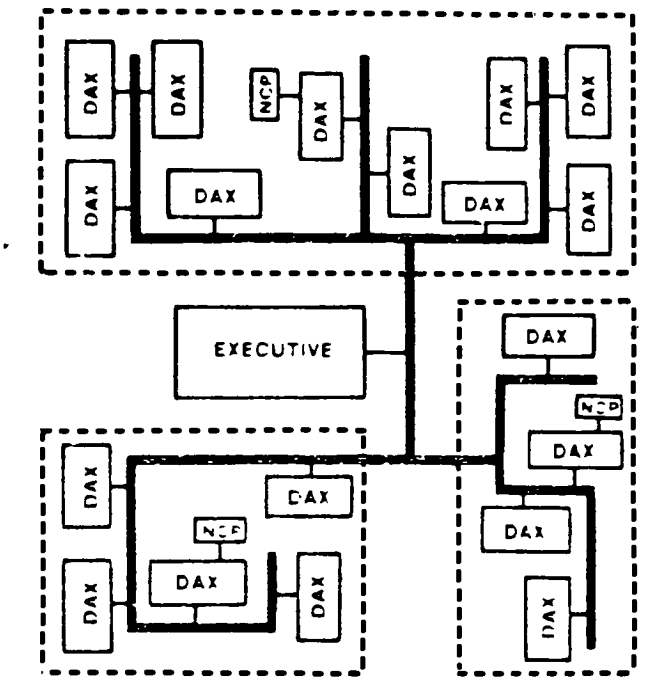


Network Control Processors

Connected to some to the DAX controllers are Network Control Processors (NCPs) which run on IBM Personal Computers (Figure 11). Each NCP supports network configuration tables, statistics/accounting files, report generation software, and Network Diagnostic Tool (NDT) software. Each NCP can be configured to monitor the entire network or just a segment of it.

Figure 11.
(Adapted from [MER 82])

Overview of CableNet showing segments of CableNet under control of NCPs.



Network Control Modules

Network Control Modules (NCMs) are programs located in each DAX. These processes communicate with NCPs for reception and transmission of network configuration tables, for event and exception reporting to the NCP, and for response to Network Diagnostic Tool (NDT) requests. These are the statistics accumulators and message passers in the network.

Network Control Consoles

Network Control Consoles (NCCs) are customer supplied terminals that access NCPs and perform NCP operations.

Network Diagnostic Tool

NDTs reside in each NCP. They allow operators to perform remote diagnostics and debugging of DAX units. Snapshots of DAX status tables can be requested.

3.2 Comments

These tools combine to form NAC. They enable NAC to keep records of physical cable/component configuration and layout, and logical cable and device configuration; to change the logical configuration of CableNet; to provide information for report generation concerning network utilization, queue lengths, traffic peaks, device errors, traffic distribution, event/exceptions to signal potential network problems, and on-line diagnostic tools

for immediate remedial action.

3.3 Case Study of Ethernet Performance

This sub-chapter summarizes a widely quoted article by Shoch and Hupp [SHO 80], which reports the performance characteristics of an Ethernet under normal, heavy, and overloaded conditions. Their object was to experimentally judge the claims made by Metcalfe and Boggs [MET 76] about Ethernet's stability, reliability, and fairness. The network monitor used in this study, is mainly a statistics gathering tool. However, it has all of the ingredients for a comprehensive network monitor according to [AME 82]: statistics gathering software, data analysis software, and artificial traffic generating capabilities. Furthermore, it demonstrates the usefulness of network monitors as an experimental testbed.

3.3.1 Test Environment

An old Ethernet installation was chosen for the measurements. It consisted of a 3Mbps, 550 meter coaxial cable, serving 120 hosts. A network monitoring program was introduced to collect statistics from one network interface running in promiscuous mode (that is, it collects every packet on the ether).

3.3.2 Performance Under Normal Conditions

The following results were reported for monitoring the network under normal conditions:

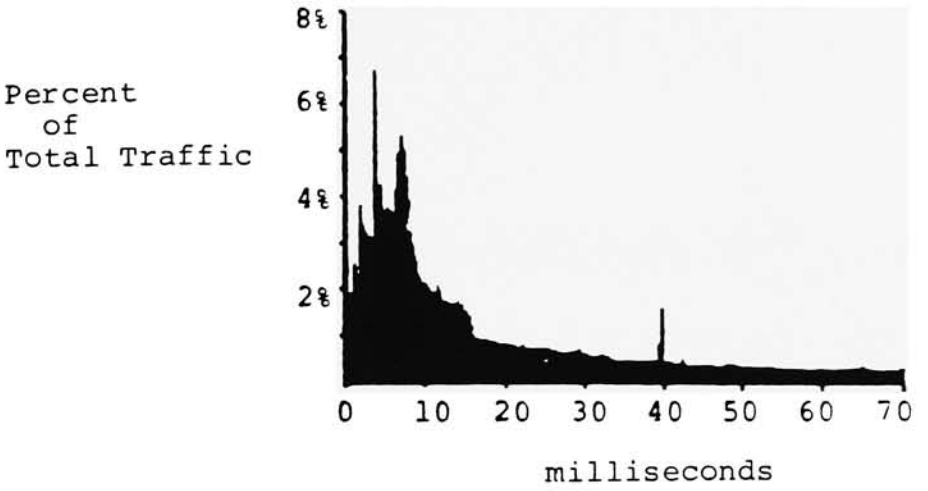
Packet errors - CRC errors occurred as low as 1 in 2,000,000 packets.

Utilization - When reported for a 24 hour day, utilization was extremely low: .84%. However, during the busiest periods utilization rose as high as 3.6% over one hour, 17% over 1 minute, and 37% in 1 second.

Packet length - As shown in Figure 9, most of the packets were small, about 32 bytes, indicating terminal traffic; however, most of the volume of traffic on the network was contained in packets greater than 512 bytes, indicating file transfers.

Inter-packet arrival times - With the propper time stamping devices, Shoch and Hupp were able to measure inter-packet arrival times. This can be a valid statistic for simulation studies, using queuing models. It is also an indication of overall network performance. Under controlled conditions, the greater the inter-packet arrival time, the worse the network is performing and conversely. As shown in Figure 12, the mean inter-packet arrival time for the network running over a 24 hour period is 39.5 ms, with a standard deviation of 55 ms. and a median of 8.5 ms..

Figure 12
(Adapted from [SHO 80])
Distribution of inter-packet arrival times.



Collisions and waiting - A small program was constructed that periodically wakes up and transmits a packet. 99.18% of all packets were sent without incidence of collision or with having to wait for the network to clear. 0.79% of the packets had to wait for the ether to free up before transmitting. Less than .3% incurred a collision.

3.3.3 Performance under high loads and overloads.

The objective of this test was to explore Metcalfe and Bogg's claims that Ethernet should continue to function very well at high loads. A high load is defined as total offered load approaching 100% channel capacity. Overload is defined as total offered load exceeding 100% channel capacity. To drive the system to high load, a special test program was placed in a number of hosts that would generate packets as long as the total offered load was less than 100% channel capacity. In order to drive the system to overload, a different version of this program was placed on each host, that would force them to attempt to transmit with impunity. The following results were noted:

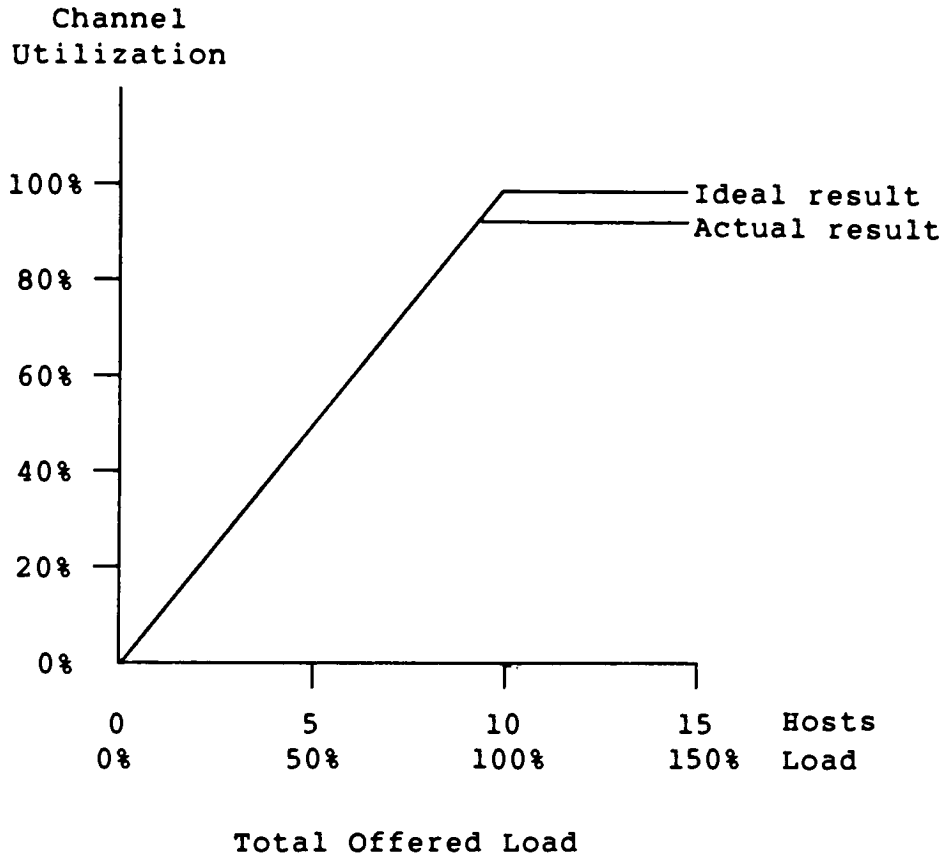
As shown in Figure 13 for the high load condition, channel utilization kept pace with offered loads from 0% to 90% of channel capacity. From 90 - 97% channel utilization flattens out, falling just 2% short of the theoretical ideal. As shown in Table 3 and Figure 14 for the overload condition, utilization increased for packets of larger byte lengths, and approached 1/e

(.37) for small packet sizes.

Table 3
(Adapted from [SHO 80]
Declining efficiency with smaller packet sizes.

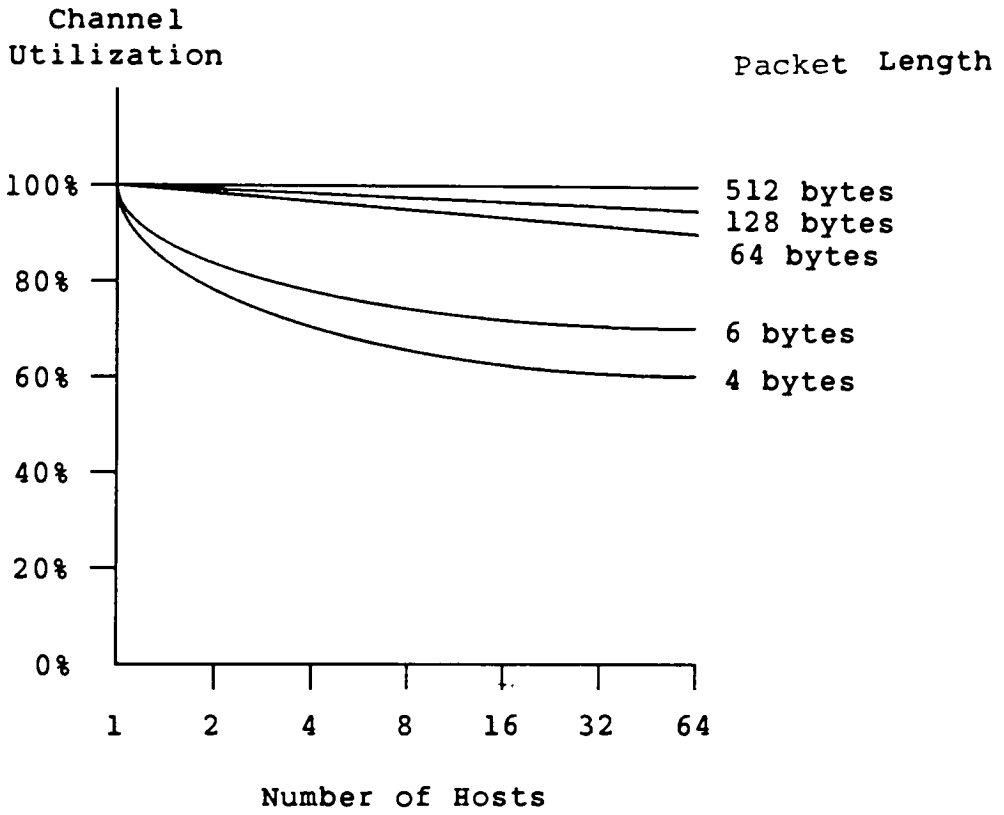
	Packet Length (bytes)				
#Hosts	512	128	64	6	4
5	96.9	95.1	94.4	71.8	----
10	96.8	91.1	88.6	68.5	58.5
32	97.2	90.2	82.6	63.5	56.2
64	97.4	91.5	84.6	60.7	54.2

Figure 13.
(Adapted from [SHO 80])
Utilization under high loads.



NOTE: Each host attempts to generate 10% load

Figure 14
(Adapted from [SHO 80])
Stability shown by utilizations
for various packet sizes under overloads.



Under high load conditions of 100% offered load (10 hosts each offering 10% of the traffic) total utilization reached 94% with each host achieving an individual throughput of 9.3% to 9.6%. Under overload conditions, with 90 hosts trying to send with impunity, total utilization again at 100%, the average throughput per host was 1.1%, ranging from .9% to 1.3%. Both cases verify that each host gets it fair share of available bandwidth and that the contention resolution algorithm designed by Metcalfe and Boggs is very fair.

Shoch and Hupp's study exemplify the usefulness of network monitors in gathering performance and behavior statistics. The generated histograms and tables could be used to determine the effect of adding or subtracting hosts from the environment. A sharp rise in collisions or other errors could indicate hardware problems. An improvement in virtual circuit protocol performance could be measured by a shorter, average inter-arrival time. A hardware upgrade from a 3 Mbps to a 10 Mbps ether will reveal its benefits in shorter inter-arrival times and a drop in the amount the cable is utilized.

Incidently, a 3 Mbps Ethernet is archaic. Most Ethernets now run at 10 Mbps. The results as shown in [SHO 80] will be similar for a 10 Mbps Ethernet because the only difference is the data rate. The specific characteristics of Ethernet such as equal access to the ether by all stations and contention resolution, do not change with a change in data rate.

3.4 Proprietary Literature

Proprietary literature provides further insight into varieties of network monitors. Many products, such as the UNIX 4.2 BSD Operating System and network controllers , are designed to use or be part of a LAN. Therefore, it is profitable for LAN exploiters to develop LAN monitoring systems to accompany their products in order to make them more attractive [OGE 81] [DAV 83]. This not only makes their products safer to purchase, but relieves customers from having to develop their own monitors.

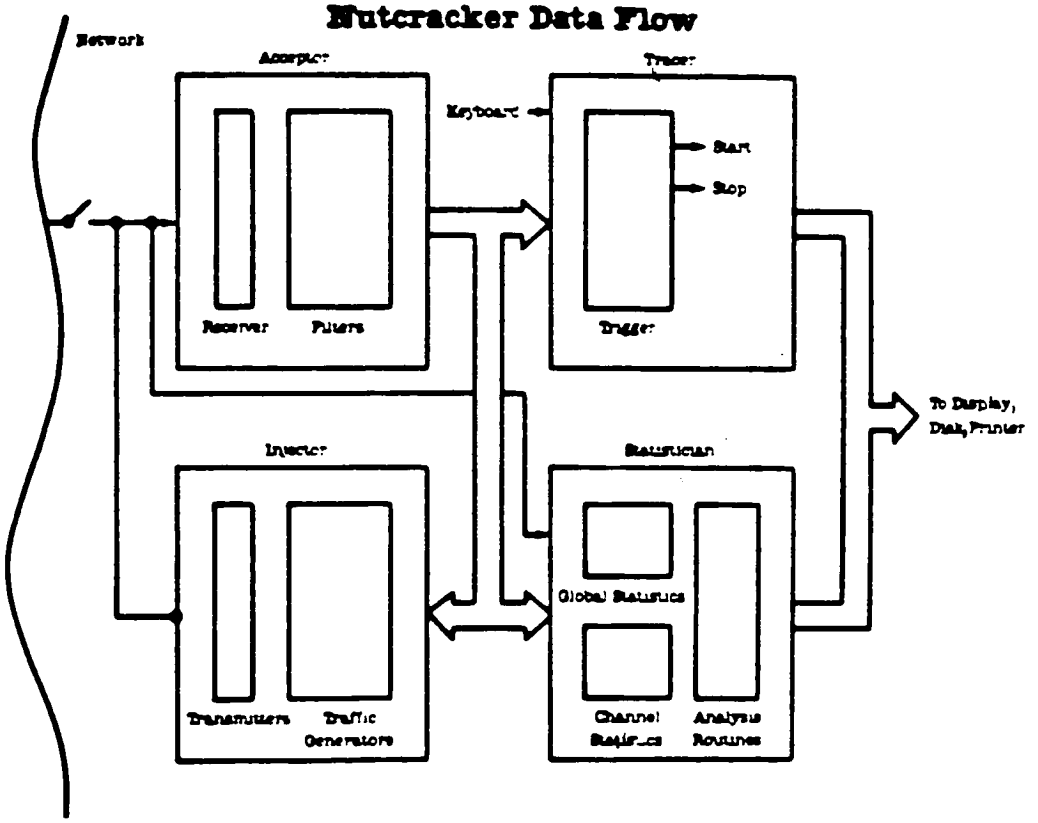
3.4.1 The Nutcracker(tm)

The Excelan Nutcracker(tm) [EXC 83] is a highly instrumented computer based network monitor. It is state of the art and currently costs between $40-50K It gathers statistics and analyzes them. Furthermore, it is intimately connected with the the Ethernet cable and keeps track of the health of network components.

As shown in Figure 15 , there are four major components to the Nutcracker(tm): Acceptor, Tracer, Injector, and Statistician.

Figure 15.
(Adapted from [EXE 83])

Four major components of the EXCELAN NUTCRACKER: Acceptor, Tracer, Injector, and Statistician.



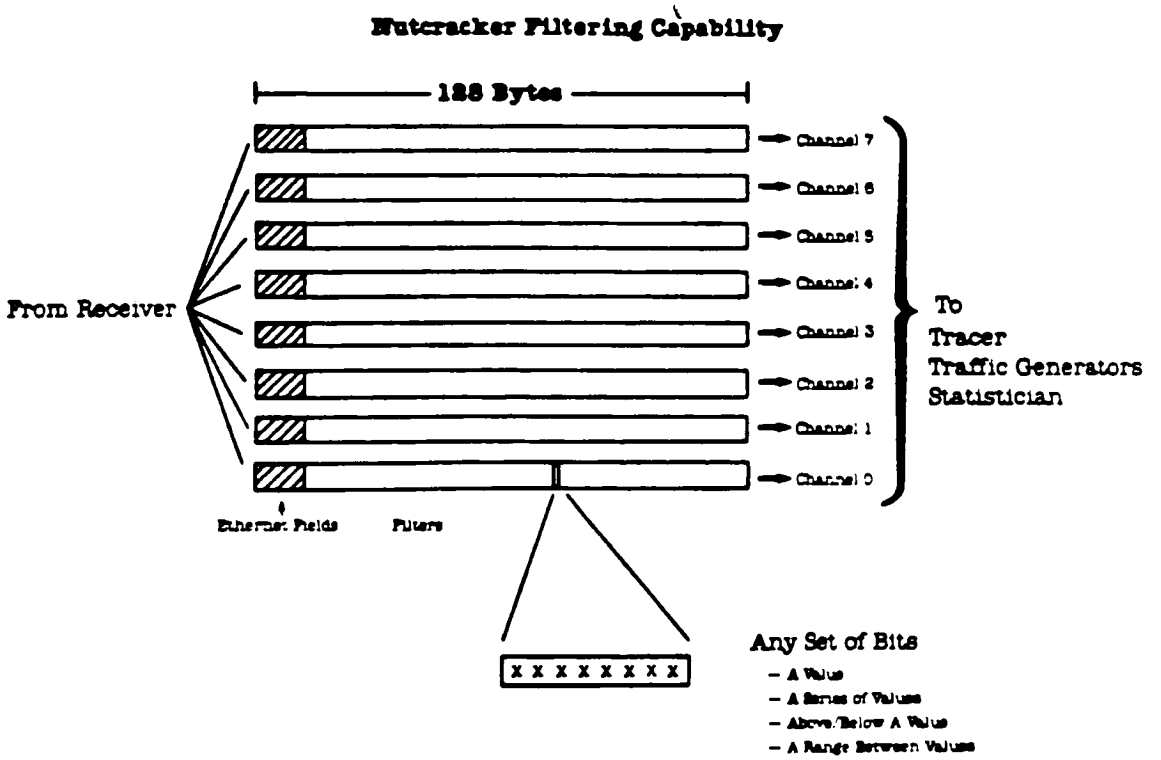
The Acceptor controls the packet acceptance process through its Receiver and the Filters. The Receiver detects:

- Out of frequency range
- Checksum errors
- Alignment Errors
- Runt packets

It can also time stamp every packet for further performance analysis. The Filters, Figure 16, are at the heart of the Nutcracker's analytical ability. Eight of them run in parallel and can be programmed separately. They allow the user to pick subsets of the Ethernet packet stream for further analysis. Traffic from or to an individual node can be selectively observed, as well as conversations between nodes or peer layers of protocols.

Figure 16.
(Adapted from [EXE 83])

The filters of the EXCELAN NUTCRACKER which allow it to keep up with high data rates, save data for further experimentation, and to trace packets.



The Tracer is used mainly to control the functions of triggering and tracing. Upon examining filter output, the tracer decides whether or not to fire a start or stop trigger, which will begin or end the collection of data from selected channels. The user can specify circular buffers to collect the data being traced, then examine or store this data. This capability provides detailed information about the behavior of specific network hardware and software components.

The Injector contains the Traffic Generators and the Transmitters. The Traffic Generators can run simultaneously and can simulate the behavior of 1-100 unique nodes, each having unique traffic generating characteristics. They can be used to drive the network to saturation for an indefinite period. These properties are important for testing and for simulating the effect that future modifications will have on the LAN. The Transmitters are associated with the Traffic Generators for transmitting clean packets or packets with user defined faults such as:

- CRC errors
- Alignment errors
- Runt Packets
- Collisions

This is advantageous when testing the ability of the various network components to handle these errors.

The Statistician accumulates and analyzes network statistics. It totals various types of error and conducts analysis,

such as optimal packet length, network utilization, average packet length, and average inter-packet arrival time. These types of statistics lend themselves to both theoretical research and network fine tuning.

The Excelan Nutcracker is a prime example of a centralized network monitor. Furthermore, it can act as traffic generator. This one unit satisfies all of the requirements for a complete network measurement center set forth in [AME 82]. It is an understatement to say that the Excelan Nutcracker can adequately monitor the performance of a LAN, test new LAN components, and conduct research.

3.4.2 NETMAN(tm)

INTERLAN Corp. markets a far less sophisticated network performance monitor called NETMAN, for "Network Management Utility" [INT 83]. NETMAN is designed to report network protocol behavior at ITP levels 0-2 (ISO layers: Physical, Data Link, Network, and Transport [SHA 83]). It is menu driven, and runs on DEC vt100 compatible terminals. It does very little statistical analysis, and is tied to gathering Physical Layer Statistics solely from Interlan controllers. These limitations aside, NETMAN presents a useful variety of data in a clear format. A summary of menus follow.

MAIN MENU DISPLAY

presents the user with a choice of menus for viewing LAN performance.

General ITP Statistics Menu

deals mainly with the routing of datagrams between networks. Numbers of packets and bytes transmitted and received are among the information presented.

Socket Use Statistics Menu

shows the number of inbound and outbound packets through every allocated socket.

Local Connection Performance Menu

shows performance of all open local sequence packet protocol connections. This includes the number of packets given to and received from the router, the number of probes given to and received from the router.

Local Connection Utilization Menu

shows the overhead introduced by sequencing packets. It gives the user a picture of how each connection is being utilized. For each socket it reports, percentages of transmitted and received data, probes, duplicates and acknowledgments are reported.

Specific Connection Statistics Menu

reports all pertinent statistics for a specific, user specified sequence packet protocol connection. Totals and percentages for packet transmissions are displayed.

Network Driver Statistics Menu

shows all the statistics maintained by an INTERLAN NI1010 Controller. These include the physical address of the controller, number of frames received, number of frames transmitted, excess collisions, runt packets, multicast frames accepted, multicast frames rejected, frames with CRC error, frames with alignment error, number of collisions, out of window collisions, module id, and the number of the firmware version running on the controller.

REMOTE STATISTICS MENU

makes available versions of the above menus at a user specified remote host.

NETMAN is useful in its ability to monitor the network from a single terminal. It is basically a reporting tool. It is not as powerful as the Excelan Nutcracker, mainly because it is limited by the hardware available to it, and because it does not analyze what it collects.

3.5 Summary

This chapter discussed several approaches to monitoring network performance and behavior. The LAN MEASUREMENT CENTER for NBSNET [AME 82] discussed several ingredients needed for a comprehensive LAN monitor: a statistics collection system, statistics analysis system, and an artificial traffic generator. Three design strategies were considered for network monitors: centralized, decentralized, and hybrid.

The Network and Administration Control System [MAT 82] was presented to show how network monitors could be used to keep a watchful eye over network utilization, device errors, traffic distribution, and provide on-line diagnostic tools to keep the network running.

The network monitor described in Shoch and Hupp [SHO 80] demonstrated the value of network monitors in LAN experimentation. In this article, they validated the claims about Ethernet made in [MET 76].

The Excelan Nutcracker [EXE 83] is a comprehensive LAN monitor, compressed into a single LAN node. It has sophisticated hardware that can both saturate a LAN and keep up with it. It can listen in on node-to-node conversations, gather and analyze performance metrics.

NETMAN [INT 83] is a network monitor that concerns itself with statistics gathering at the controller and socket levels. Although it does not analyze the data, it does provide a global

view of network activity.

4 TOOLSNET ANALYSIS PACKAGE (TAP)

The previous material laid the foundation for the kinds of performance metrics and design strategies that I have implemented in TOOLSNET ANALYSIS PACKAGE (TAP). TOOLSNET is a local area network that serves the Compugraphic Corporation's Engineering community, at Corporate Headquarters in Wilmington Massachussetts.

4.1 The TOOLSNET Environment

TOOLSNET usually serves eight host systems--all of them in the same room: 3 PDP-11/70s (systems D, F, and H) and 3 VAX 11/750's (systems X V X, and sometimes S) and one VAX 11/785 (system Z). The VAXES run the UNIX(tm) Version 4.2 Operating System and the 11/70s run the UNIX(tm) Version 7 Operating System. The protocols for the TOOLSNET local area network, were written in 1983 by a Compugraphic employee. The TCP/IP protocols that normally come with UNIX(tm) 4.2 could not be added to the PDP 11/70's because their operating systems could not be expanded enough to contain them. Furthermore, TCP/IP does not provide an inter-machine, virtual file system as does TOOLSNET. The TOOLSNET cable length is at most, 200 meters. which means that the degradation in performance due to propagation delay is so small that it is negligible to worry about it.

There are three types of packets: data, acknowledgement (ACK), and negative acknowledgement (NACK) (see [TAN 81]). The acknowledgement packet scheme is overly simplistic. Every data packet received is acknowledged (ACKED) immediately by the driver. What makes the scheme simplistic is that the driver simply changes the type field in the data packet to ACK and ships the entire packet back to its source. This results in ACK packets ranging from the minimum packet size (64 bytes) to the MAXIMUM packet size (1518 bytes). Although this wastes space, it is very fast. Therefore, acknowledgement packets effectually mirror their data packets and double the amount of bandwidth that is used up.

Prior to TAP a small program gathered the statistics collected by the driver and wrote them to the user's screen. Here are the statistics it gathered:

physical address of the controller,
number of frames received,
number of frames transmitted,
excess collisions,
runt packets,
multicast frames accepted,
multicast frames rejected,
frames with CRC error,
frames with alignment error,
number of collisions,
out of window collisions,
module id,
and the number of the firmware version of the controller.

While this small program provided a model for interfacing with the TOOLSNET driver, some important statistics remained to be gathered, such as packet length and packet number counters that are needed to calculate mean packet length.

4.2 Methodology

Under the constraints of designing a network monitor for a research and development facility it was only possible to follow a distributed approach [AME 82]. A centralized or hybrid approach to statistics gathering that would include a controller

running in promiscuous mode was not feasible. Each host on TOOLSNET contains statistics gathering software that periodically reads statistics from the driver and appends them to holding files. Performance calculations (for example, utilization) are performed by a central program which runs on one VAX. These can be done whenever desired, but usually after hours. This program polls each system for the contents of their holding files, from which global utilization statistics are calculated. Each line in the holding files includes all of statistics mentioned above, each time that the statistics were collected from the driver, and the number of packets and bytes that the system being polled sent to each of the other TOOLSNET hosts. A few sample lines with headings from one of these holding files are shown in Table 4. In order for someone to have TAP do performance calculations on demand, he must be logged onto the central VAX and execute the program from there.

The output from the central program is also saved in holding files for processing by another program which tallies up TOOLSNET performance and behavior for an entire 5 day work week. The reader is referred to Appendices A, B, and E for a more detailed overview of the design, manual pages, and the code of TAP.

Table 4
Contents of Holding Files

Key:
rf = received frames, xf = transmitted frames, ec = excess collisions, rc = recieved collisions,
lf = lost frames, ce = checksum error, ae = alignment errors, co = collisions,
[DFHVWXZS]n = Number of packets from system [*],
[DFHVWXZS]l = Total number of bytes (packet length) from system [*].

time	rf	xf	ec	rc	lf	ce	ae	co	Dn	Dl	Fn	Fl	Hn	Hl	Vn	Vl	Wn	Wl	Xn	Xl	Zn	Zl	Sn	Sl
Sep 12 16:01:02	2766	2765	0	0	D	1	1	0	1692	344788	0	0	36	3604	964	115176	0	0	0	0	74	7512	0	0
Sep 12 16:06:01	951	951	0	0	0	0	0	0	498	134152	0	0	0	0	452	46060	0	0	0	0	0	0	0	0
Sep 12 16:11:01	434	434	0	0	0	0	0	0	364	78364	0	0	0	0	70	9348	0	0	0	0	0	0	0	0
Sep 12 16:16:01	1283	1283	0	0	0	1	1	0	1260	250732	0	0	0	0	22	2220	0	0	0	0	0	0	0	0
Sep 12 16:21:01	3630	3631	0	1	0	0	0	1	3592	1137804	0	0	0	0	D	0	0	0	0	0	38	8844	0	0
Sep 12 16:26:02	1794	1793	0	0	0	1	1	0	1794	554916	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sep 12 16:31:01	7313	7313	0	2	0	2	2	4	7180	2215240	0	0	132	13280	0	0	0	0	0	0	0	0	0	0
Sep 12 16:36:01	5714	5714	0	1	0	1	1	1	5560	1854332	0	0	0	0	0	0	0	0	10	1100	142	23336	0	0
Sep 12 16:41:01	1168	1168	0	0	0	0	0	0	448	123328	714	94084	0	0	0	0	0	0	0	0	6	600	0	0
Sep 12 16:46:01	1643	1643	0	2	0	0	0	2	0	0	1604	215208	0	0	0	0	0	0	0	0	40	9120	0	0
Sep 12 16:51:01	1101	1101	0	0	0	0	0	0	0	0	1100	138404	0	0	0	0	0	0	0	0	0	0	0	0
Sep 12 16:56:02	1102	1102	0	0	0	0	0	0	0	0	1102	133460	0	0	0	0	0	0	0	0	0	0	0	0

The scope of statistics that TAP can gather is naturally limited by the available hardware. The lack of sophisticated timing devices, for time stamping packets in between layers [MUR 84] and the lack of synchronization among host clocks (for end-to-end delivery times) cancels the ability of TAP to measure delay. However, TAP measures the group of utilization metrics discussed in [STA 84], [TAN 81], and [ALM 79]. Specifically, TAP calculates channel efficiency (eq. 4, eq. 9, and eq. 10), asymptotic throughput efficiency (eq. 5), relative load (eq. 11), perceived efficiency (eq. 12), and average response (eq. 13). For a comprehensive overview of the design and implementation of TAP, you are referred to Appendix A.

Calculating utilization is an small part of TAP. A far more important function is the fact that it is always running and monitoring TOOLSNET. The holding files created by the statistics gathering program on each system provide snapshots of the driver statistics mentioned above, for the last 5 minutes, last hour, or from the beginning of the working day. TOOLSNET behavior at the driver level is constantly tracked and can be called up on demand; see Appendix B for the manual pages of shell scripts and programs that have been written for viewing TAP output. Sample output from these scripts is also provided in Appendix B.

So far TAP has been able to pinpoint both crippled printer servers that cannot accept the packets that are sent to them over and over again and potential hardware problems. By observing traffic through TAP, the System Administrators in the TOOLS GROUP

have gained more understanding of what the TOOLSNET protocols do. For example, they noted that one host on TOOLSNET uses 20-30 packets of 100 bytes each to list a directory on a remote host. This has given them more motivation to reimplement TOOLSNET. If systems are slow, TAP relieves System Administrators from guessing whether or not trouble or heavy traffic on the network is causing the slow down. A lot of TOOLSNET activity on any host causes a notable decline in its performance. This is because TOOLSNET runs at a very high priority, there is only an available pool of 30 packets, and processes go to sleep, indefinitely, while they wait for packets (still more motivation to reimplement TOOLSNET). Furthermore, administrative decisions about relocating project groups to other computers so that they generate less traffic on TOOLSNET have been supported through the raw data collected by TAP.

4.3 Results

After a week of monitoring TOOLSNET from 8 a.m. - 5 p.m., Monday through Friday, the following characteristics were noted:

4.3.1 Overall Traffic Characteristics

4.3.1.1 Totals

The total number of packets and bytes generated by each system is displayed in Table 5.

Table 5

Total Packets and Bytes per System

SYSTEM ID	TYPE COMPUTER	PACKETS x 1000	BYTES x 1000
D	PDP 11/70	587.4	122,226.4
F	PDP 11/70	775.0	169,785.5
H	PDP 11/70	624.7	111,950.9
S	VAX 11/750	412.3	49,466.7
V	VAX 11/750	379.9	77,139.8
W	VAX 11/750	297.0	48,654.5
X	VAX 11/750	409.1	45,729.6
Z	VAX 11/785	643.3	98,717.6

ACK packets generated by the TOOLSNET protocols directly reflect the data packets which they acknowledge. Consequently, the high number of bytes generated by the the PDP-11/70s exhibits the acknowledgement of large packets, which would be typical of file traffic being sent to line printers. System Z has the highest number of packets and bytes transmitted for any VAX; this also confirms the weight of printer traffic because System Z supports the only laser printer in TOOLSNET.

4.3.1.2 Errors

The errors noted on TOOLSNET are reported in Table 6.

Table 6

TOTAL ERRORS ON TOOLSNET FOR ONE WEEK

EC	RC	LF	CE	AE	CO
0	738	0	1870	1806	409

*Where EC = excess collisions, RC = received collisions, LF = lost frames, CE = checksum errors, AE = alignment errors, and CO = collisions.

The total number of packets transmitted during the test week was 4,130,478 giving a total error frequency of 0.0012.

4.3.2 Utilization Statistics

Mean packet length and utilization statistics were calculated on a day to day. Utilization according to [TAN 81] (eq. 7) is displayed in Table 7. Utilization according to [ALM 79] is displayed in Table 8 (eqs. 1, 2, 3, 4). The other utilization statistics mentioned in [ALM 79] are displayed in Table 9 (eqs. 8, 9, 10).

The constants, which are characteristics of TOOLSNET, involved in the calculations for Table 7 were: C = 10 Mbps, tw = .000001, e = 2.7, mean packet length is given in the table for

each day. Eq. 10 is:

$$\frac{P/C}{P/C + 2*tw*e}$$

Table 7

Utilization with number of contention slots set to 'e'.

Measurement	MON	TUE	WED	THU	FRI
P in bytes	189	165	178	187	198
UTILIZATION	1.037	1.043	1.039	1.039	1.035

The utilization of TOOLSNET is slightly greater than utilization reported in [SHO 80], for comparable period of time.

Equation 4 is:

$$\frac{P/C}{P/C \quad + \quad 2*tw*Z}$$

Table 8

Utilization from with a variable number of contention slots.

Measurements	MON	TUE	WED	THU	FRI
P in bytes	189	165	178	187	198
Q=1, A=1, Z=0, U=	1	1	1	1	1
Q=2, A=.5, Z=1, U=	0.96	0.95	0.96	0.97	0.97
Q=3, A=.44, Z=1.25, U=	0.96	0.95	0.96	0.96	0.96
Q=4, A=.42, Z=1.37, U=	0.95	0.94	0.96	0.95	0.96
Q=5, A=.41, Z=1.44, U=	0.95	0.94	0.95	0.95	0.95
Q=6, A=.40, Z=1.48, U=	0.95	0.94	0.95	0.95	0.95
Q=7, A=.39, Z=1.52, U=	0.95	0.94	0.95	0.95	0.95
Q=8, A=.38, Z=1.56, U=	0.95	0.94	0.95	0.95	0.95

Table 9 was calculated with the following variables: Q = 1000, A = 0.37, Z = 1.703. Recall the equations for offered load, perceived efficiency, and average response time are:

$$RL = Ro / U \qquad (11)$$

$$PE = 1 - RL. \qquad (12)$$

$$AR = P / C / PE \qquad (13)$$

Table 9

	Measurement				
P in bytes	189	165	178	187	198
Asymptotic Utilization	0.94	0.94	0.94	0.94	0.94
Offered load Ro	0.003	0.003	0.005	0.003	0.01
Relative Load	0.005	0.005	0.005	0.003	0.01
Perceived Efficiency	0.99	0.99	0.99	0.99	0.99
Average Response Time (in seconds)	0.0002	0.0002	0.0002	0.0002	0.0002

4.3.3 Source to destination Traffic

Source/destination and error traffic was measured on a day to day basis and then totaled at the end of the week for each system. Table 10 shows the number of packets and bytes received by each system on TOOLSNET.

Table 10
Number of packets sent to
each system on TOOLSNET for one week.

ey:

DFHVWXZS]n = Number of packets to system [*] in 1000s
DFHVWXZS]1 = Total number of bytes (packet length) to system [*] in 1000s

ay	Dn	D1	Fn	F1	Hn	H1	Vn	V1	Wn	W1	Xn	X1	Zn	Z1	Sn	S1
on	89.3	20808.6	132.8	25773.7	110.8	14310.8	129.8	29220.7	40.5	5210.1	43.3	6476.7	107.7	17757.0	222.6	27236.2
ue	85.1	15278.1	188.9	30746.8	157.1	24221.8	47.5	7147.5	37.5	6926.3	59.7	8716.7	106.5	18632.7	159.9	17657.3
ed	220.0	44392.2	217.6	39565.6	181.2	38836.7	127.0	26520.5	78.6	15164.0	133.3	7962.7	191.9	32406.0	10.2	1914.7
hu	115.6	22644.1	133.5	42737.8	91.4	15639.8	45.4	10173.5	28.4	7655.4	95.9	12253.9	130.6	16339.5	15.1	2141.1
ri	77.2	19103.4	102.2	30961.6	84.1	18941.9	30.3	4077.6	112.0	13698.6	76.8	10319.6	106.6	13582.3	4.6	517.5
OTAL	587.4	122226.4	775.0	169785.5	624.7	111950.9	379.9	77139.8	297.0	48654.5	409.1	45729.6	643.3	98717.6	412.3	49466.7

4.4 Validation

The traffic, packet length, and error statistics were very similar to TOOLSNET behavior when monitored by an EXCELAN NUT-CRACKER, that was borrowed from another department.

4.5 Comments

The values obtained for the family of utilization statistics confirm the fact that Ethernet Lan Channels usually experience a low percentage of used up bandwidth [SHO 80], [STA 84]. Of even more practical importance to understanding the TOOLSNET environment are the raw statistics and their cumulative totals, when processed by the shell scripts, and programs mentioned in Appendix B.

5 Personal Experiences in Developing TAP

This chapter was written on behalf of others who wish to tread on the path of developing a network monitoring system. The items in this chapter were things that I learned as I developed and implemented TAP and that are not obvious from the preceding chapters.

5.1 Operations procedures

One of the most terrifying experiences was learning to shut-down and resurrect three different kinds of computer: PDP-11/70, VAX 11/780, and VAX 11/785. For an applications software addict like myself, the realization that I could corrupt a file system or destroy other peoples' work, caused me to be extremely cautious. I am grateful for a patient operation's crew.

5.2 Device Drivers and IOCTLs

The TOOLSNET driver was a garden variety UNIX device driver, of which I had absolutely no idea. If you have just a rudimentary knowledge of device drivers I recommend obtaining a copy of "Writing a UNIX Driver" by MASSCOMP or some other book that holds your hand through Unix drivers and how they fit in with the UNIX

Operating System.

It also became apparent that I had to learn about ioctls in order to turn source and destination counters on/or off (Appendix B).

A more than casual understanding of block io movement through system calls such as: iomove, uiomove, and bcopy, would also have come in handily.

5.3 Lousy Code

Be prepared to have to struggle through undocumented, cryptic, and sometimes childish code if you decide to add a network monitor to a LAN. Furthermore, do not trust the code to which yours must interface. A slightly improper format to an iomove call added 3 weeks to this project.

5.4 Layered Protocols

Although no network precisely follows the ISO-OSI seven layered approach, having a good understanding of this approach to organizing a network was a great help to understanding how data was moved from user to controller and visa-versa, through the TOOLSNET protocols.

6 Recommendations for the Future

Additions and improvements to TAP should take on the following forms:

6.1 Accounting

Accounting elements should be built into TAP that will enable the TOOLS GROUP to monitor TOOLSNET use by groups and by users by tracking the UID's and GID's (user and group identification numbers) resident in each packet. This addition is coming soon at the request of the management, who feel that it will help them to pinpoint TOOLSNET abusers, and to find out which project groups generate a lot of traffic on TOOLSNET and move them, if possible.

6.2 Packet Length Counters

It would be nice to insert counters in the driver which record the frequency of packets for a given length. This was avoided this time around due to the fact that there is no room left in the 11/70 kernels for them. These counters would help us keep track of the characteristics of the traffic on the net. For example, a large number of small packets would indicate terminal

intensive traffic, whereas a large number of large packets would indicate file intensive traffic.

6.3 Addition of Time Stamping Devices

Should CG strike oil in the parking lot, or better yet, land a huge Defense Contract and have money to squander, the purchase of sophisticated time stamping devices would be an asset to calculating delay statistics. I feel that the true performance calculations lie in delay, rather than in utilization. Time stamping packets in-between protocols could lend a true sense of protocol performance and end-to-end character delivery delays.

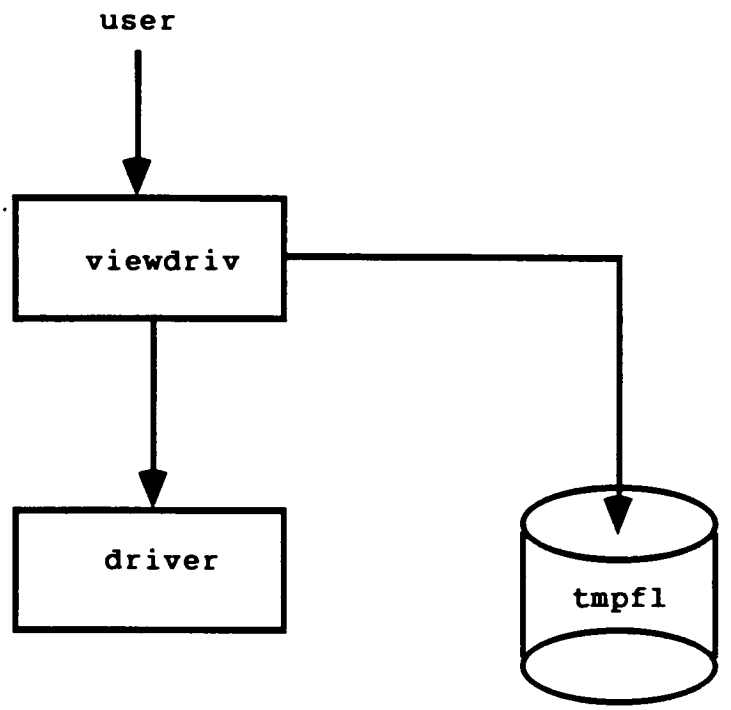
APPENDIX A

Design of TAP

This appendix describes the programs that were written to comprise TAP. Each program is then followed by its high-level structured diagram. Additional documentation concerning the structures of the files that are maintained by TAP is presented in the manual pages of Appendix B.

A.1 Viewdriv

Viewdriv allows the user to view the totals held in the driver's counters. It will also save these totals into a temporary file (tmpfl) that is read by gatherstats (below). This avoids losing these totals, because every time viewdriv is invoked the counters are cleared. Sample output is in Appendix B.



SD1

Structured diagram of viewdriv. A program that views driver's counters on demand from user.

A.2 Gatherstats

Gatherstats will be invoked automatically on a daily basis by cron. It will run every 5 minutes, on each host. It's main purpose is to invoke the TOOLSNET driver to retrieve statistics from the controller, and to append these statistics, to two files.

Specifically, gatherstats has three sources of input:

- The stats (if any) held in tmpfl (from viewdriv invocations)
- Any stats collected by the driver
- A file called, blkrecentread, which contains the totals from the controller over the last hour.

Gatherstats has four output files:

- Dayread: the continuously appended file that shows the totals collected by the driver for every gatherstats invocation for that day. Dayread is in CG-DBMS format to afford users the ability to use data base tools on it.
- Blkdayread: the binary version of dayread, which is used by doglobals for better performance by doing block reads, rather than reading in files line by line that are formated for our data base package at CG
- Recentread same file as blkrecentread, but in CG-DBMS format
- Blkrecentread, is also used as an output file as explained below

The actual behavior of this program is to:

1. Open tmpfl and read it

2. Get statistics from the driver

3. Add 1 to 2

4. Read in blkrecentread

    If there is greater than say, an hour's worth of

    readings in there

        Then

            get rid of the first record (line 1)

append 3 to contents of blkrecentread

append 3 to contents of recentread

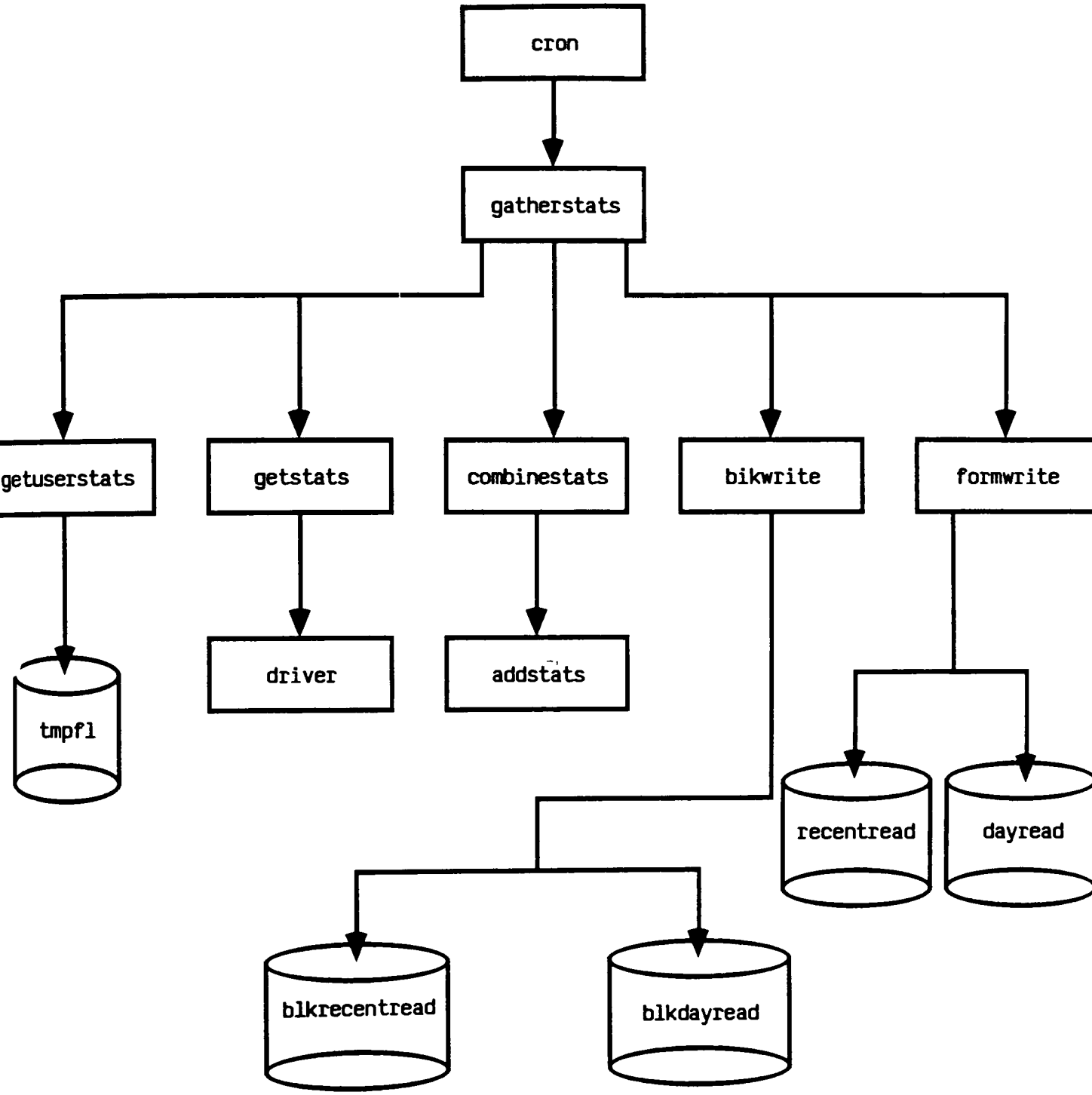
Else

just append 3 to contents of blkrecentread

just append 3 to contents of recentread

5. Write over the entire contents of recentread with the new contents.

6. Append 3 to blkdayread and dayread.



SD2

Structured diagram of gatherstats. The statistics collecting program that runs on each host.

A.3 Doglobals

Doglobals is the centerpiece of statistical performance calculations. It resides on one VAX. It is mainly responsible for calculating the global utilization measurements mentioned in Chapter 2. The statistics it calculates are mentioned below.

It behaves in the following way:

1) Doglobals is executed by the user with a "d" or an "r" switch. The "d" switch tells doglobals to obtain its raw measurements from the blkdayread files from each host. The "r" switch tells doglobals to use the blkrecentread files from each host. Remember "blkdayread" and "blkrecentread" are produced by the gatherstats program that runs on each host.

2) Doglobals then finds out the names of the hosts available for processing by reading a file called "h_avail". The line in "h_avail" might look like this:

    DFHVWXZ

3) For each host represented in h_avail, doglobals reads the blkdayread or blkrecentread files and calculates the mean packet length, offered load, transmission and reception rates detected by that particular host.

4) It also totals up the traffic seen by each host and for the entire network for that given day (in which case it is

accessing blkdayread) or period (in which cases it is accessing blkrecentread).

5) After each system has been visited, doglobals calculates the global network performance statistics for TOOLSNET, such as:

overall mean packet length,

total offered load,

utilization according to [TAN 81] (eq. 9),

utilization according to [STA 84] (eq. 10),

utilization according to [ALM 79] (eq. 4),

hypothetical asymptotic throughput efficiency (eq. 5),

relative load (eq. 8),

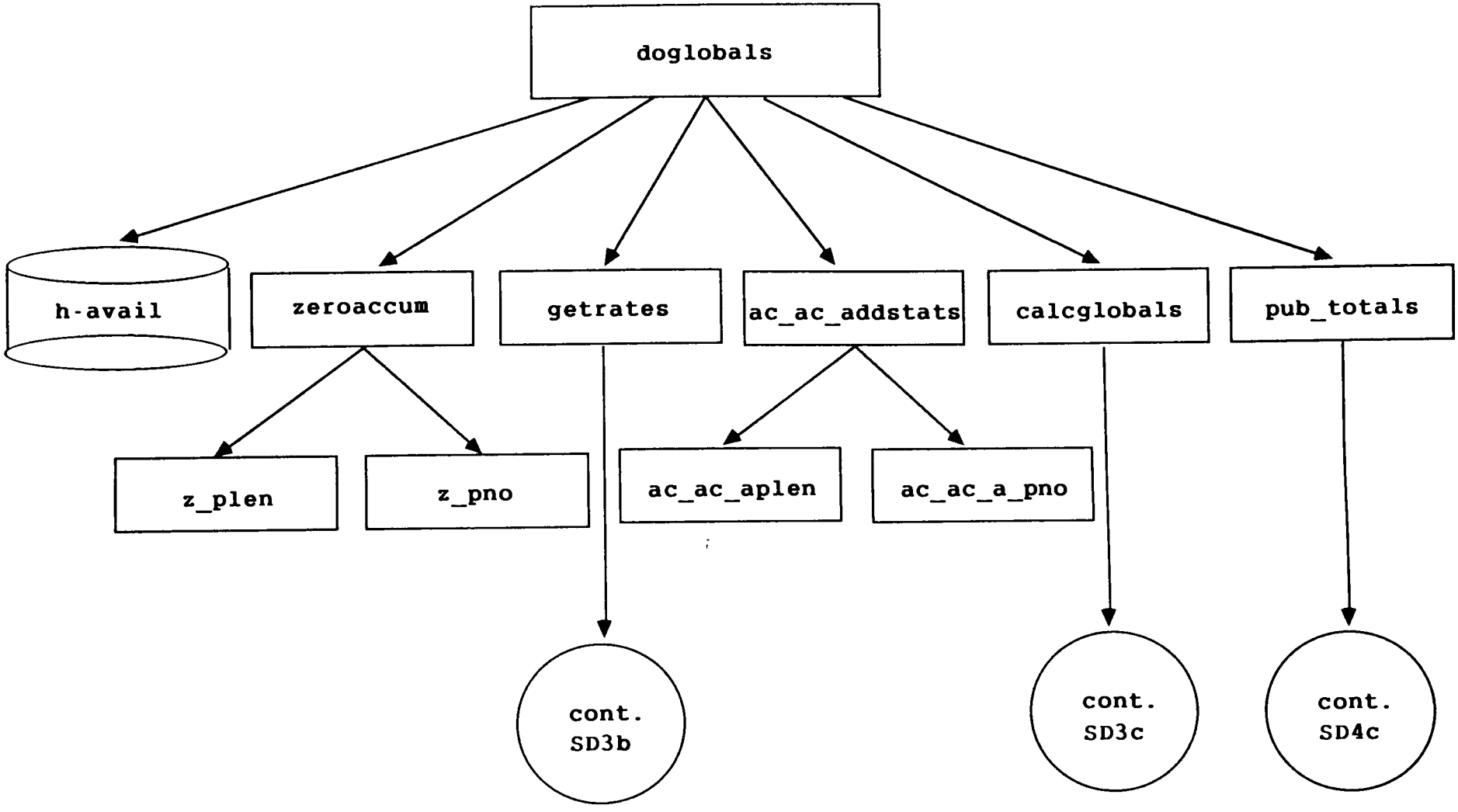
perceived efficiency (eq. 11),

and

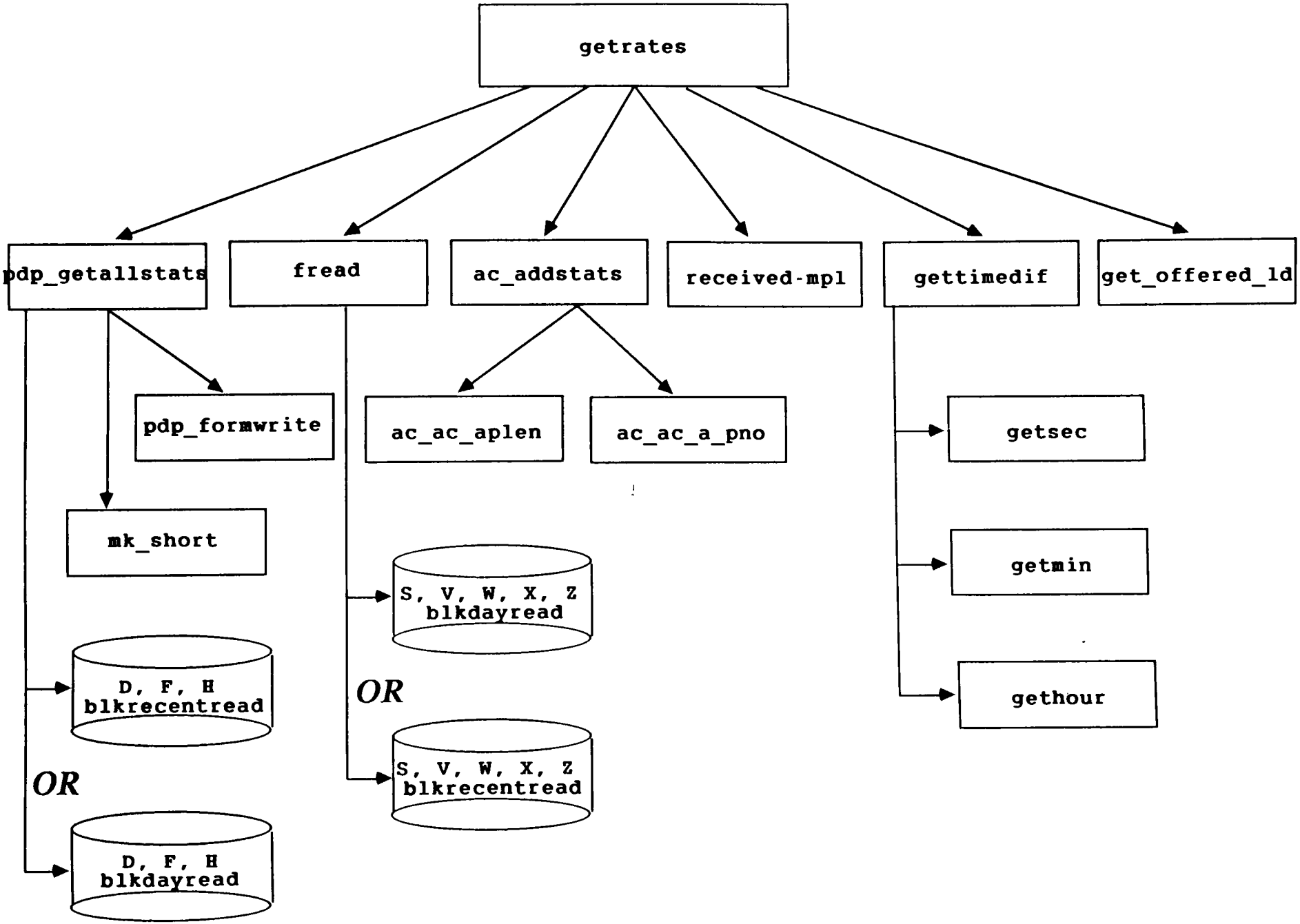
average response time (eq. 12).

6) If the "d" switch was selected by the user (usually cron at 6:00 AM) the system and network totals are dumped out to holding files for viewing by people and for processing by yet another program, tnet_week (see tnet_week.c).

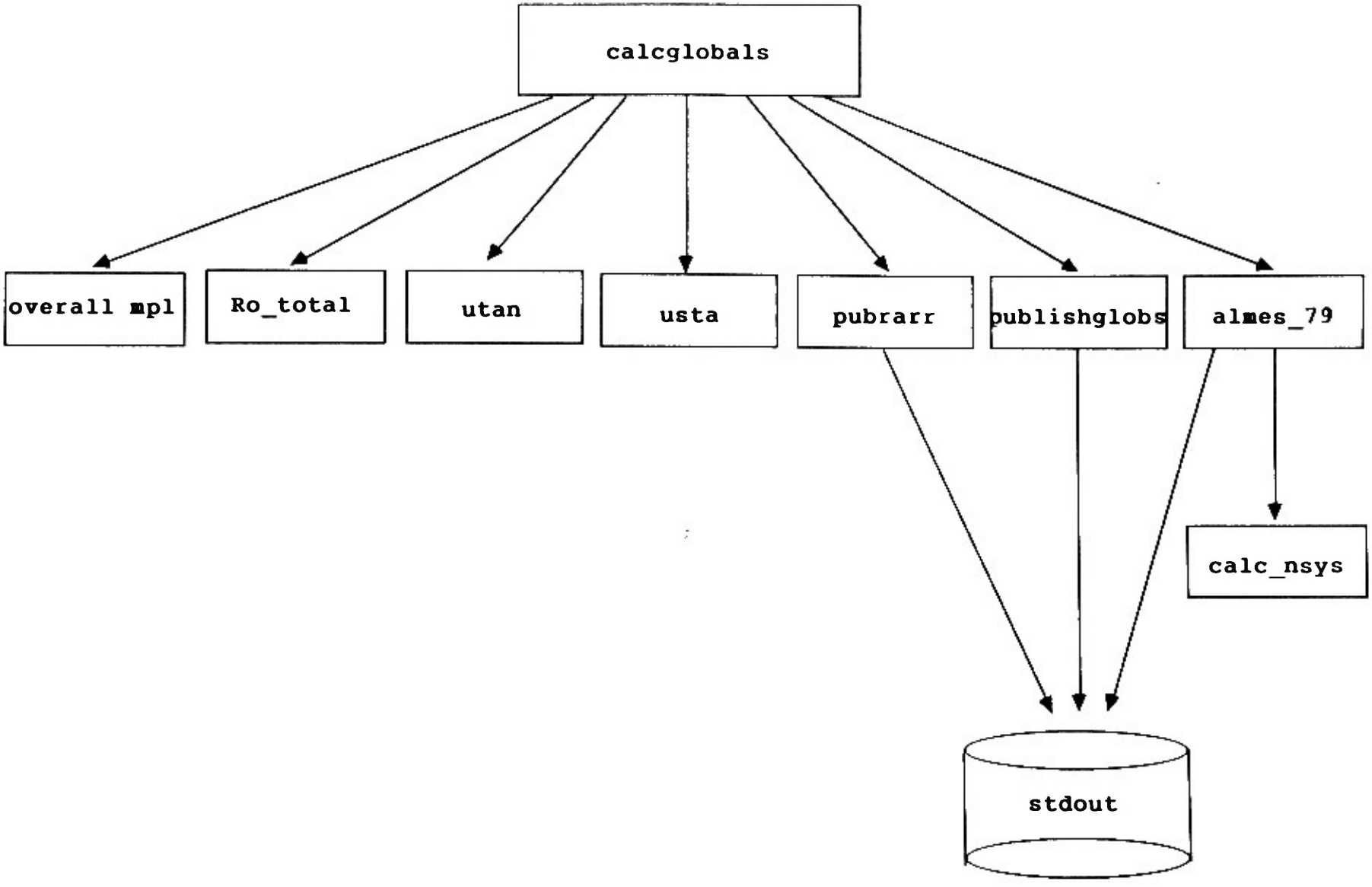
See the diagram of the procedure, pub_totals() for further documentation concerning these holding files.


doglobals
h-avail
zeroaccum
getrates
ac_ac_addstats
calcglobals
pub_totals
z_plen
z_pno
ac_ac_aplen
ac_ac_a_pno
cont.
SD3b
cont.
SD3c
cont.
SD4c

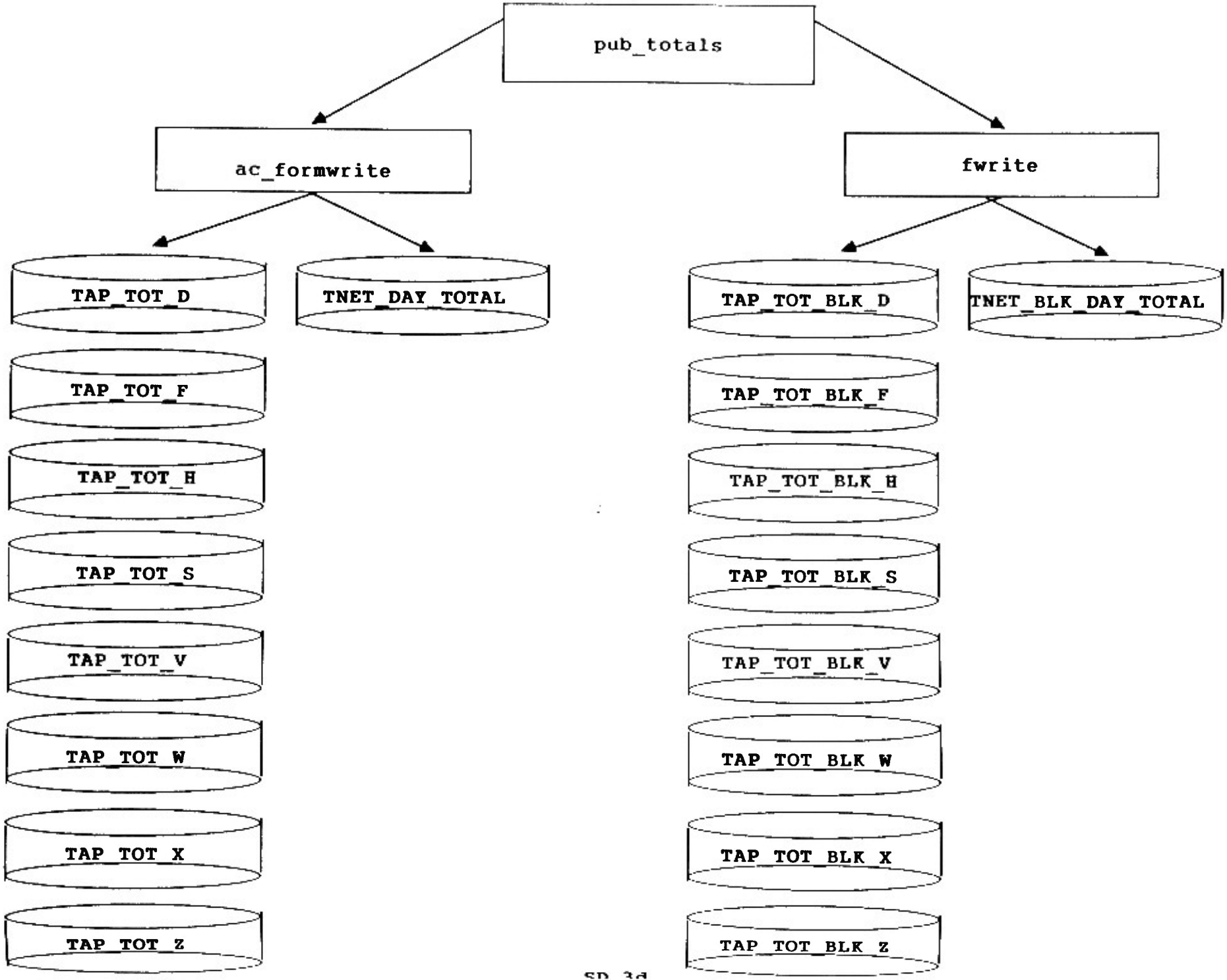
SD 3a


getrates
pdp_getallstats
fread
ac_addstats
received-mpl
gettimedif
get_offered_ld
pdp_formwrite
ac_ac_aplen
ac_ac_a_pno
getsec
mk_short
S, V, W, X, Z
blkdayread
getmin
D, F, H
blkrecentread
OR
gethour
S, V, W, X, Z
blkrecentread
OR
D, F, H
blkdayread

SD 3b


calcglobals
overall mpl
Ro_total
utan
usta
pubrarr
publishglobs
almes_79
calc_nsys
stdout

SD 3c


pub_totals
ac_formwrite
fwrite
TAP_TOT_D
TNET_DAY_TOTAL
TAP_TOT_F
TAP_TOT_H
TAP_TOT_S
TAP_TOT_V
TAP_TOT_W
TAP_TOT_X
TAP_TOT_Z
TAP_TOT_BLK_D
TNET_BLK_DAY_TOTAL
TAP_TOT_BLK_F
TAP_TOT_BLK_H
TAP_TOT_BLK_S
TAP_TOT_BLK_V
TAP_TOT_BLK_W
TAP_TOT_BLK_X
TAP_TOT_BLK_Z

SD 3d

A.4 tnet week

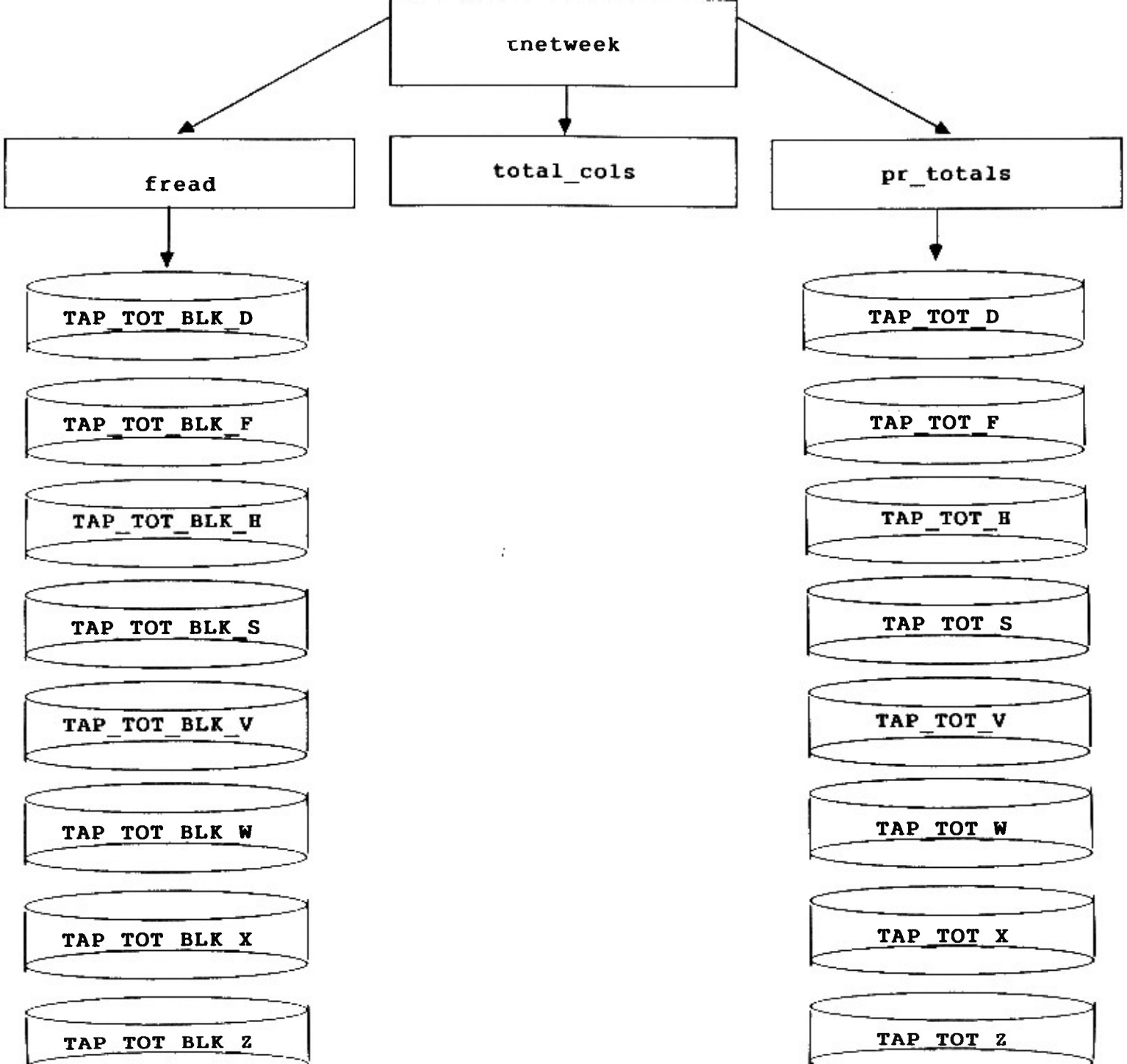
The purpose of tnet_week() is to sum up the TOOLSNET activity for each system and for TOOLSNET itself for a given work week. The procedure pub_totals() in the file doglobals.c appends a day's worth of each system's activity to the files:

```
TAP_BLK_TOT_D    ( binary block i/o files)
TAP_BLK_TOT_F
TAP_BLK_TOT_H
TAP_BLK_TOT_S
TAP_BLK_TOT_V
TAP_BLK_TOT_W
TAP_BLK_TOT_X
TAP_BLK_TOT_Z

TAP_TOT_D        (ascii versions)
TAP_TOT_F
TAP_TOT_H
TAP_TOT_S
TAP_TOT_V
TAP_TOT_W
TAP_TOT_X
TAP_TOT_Z
```

Pub_totals() also appends a days worth of TOTAL TOOLSNET activity to the files: TNET_DAY_BLK_TOTAL and TNET_DAY_TOTAL.

Tnet_week() runs at the end of the week and totals up the contents of the *BLK* files and appends the totals to the ascii non-BLK files so that people can read them.


tnetweek
fread
total_cols
pr_totals
TAP_TOT_BLK_D
TAP_TOT_BLK_F
TAP_TOT_BLK_H
TAP_TOT_BLK_S
TAP_TOT_BLK_V
TAP_TOT_BLK_W
TAP_TOT_BLK_X
TAP_TOT_BLK_Z
TAP_TOT_D
TAP_TOT_F
TAP_TOT_H
TAP_TOT_S
TAP_TOT_V
TAP_TOT_W
TAP_TOT_X
TAP_TOT_Z

A.5 Cleanup

Cleanup is a command file runs at after hours. It's major purpose is to calculate and print out TOOLSNET statistics gathered during the previous workday. It will run on only one host, with at least one back up on another. It will behave in the following manner:

- Sort each hosts' dayread file by number of transmissions and print out the sorted file. This is for observing traffic patterns on TOOLSNET.
- call doglobals and append its calculations to an archival file
- Delete dayread, blkdayread, recentread, blkrecentread and tmpfl files.

A.6 Miscellaneous programs

Several small programs have been written to process data held in the "dayread", "recentread", TAP_TOT_*, and TNET_DAY_TOTAL files mentioned above. This was the original intent for placing these files in CG-DBMS format. These programs form the most important user interface to TAP. They are not part

of the design and are not listed here; however, their manual pages and output are listed in Appendix B.

APPENDIX B

UNIX MANUAL PAGES

This appendix contains UNIX manual pages for the preceding programs and for other programs and shell scripts that were written to provide TAP users with convenient viewing of TOOLSNET activity. No structured diagrams are provided. Output from some of these programs is at the end of this appendix. The output is in the form of a terminal session with TAP, one in which the user sits at his terminal and types TAP commands. This is made possible by a UNIX program called "script", which records the system's output to a terminal in a file called "typescript". Consequently, the reader will be able to see how TAP is actually used from a terminal--a "demonstration" so to speak. The typescript file was edited to put some of it in 132 column mode and other parts of it in 80 column mode.

NAME

doglobals - do global calculations for TOOLSNET ANALYSIS PACKAGE.

SYNOPSIS

doglobals d|r

DESCRIPTION

Doglobals should run on only 1 VAX in TOOLSNET. It is the premier program for statisitical calculations for TOOLSNET ANALYSIS PACKAGE (TAP). The purpose of doglobals is to calculate global performance statistics. The statistics it calculates are mentioned below.

It behaves in the following way:

1) Doglobals is executed by the user with a "d" or an "r" switch. The "d" switch tells doglobals to obtain its raw measurements from the /usr/adm/blkdayread files from each system. The "r" switch tells doglobals to use the /usr/adm/blkrecentread files from each system. "Blkdayread" and "blkrecentread" are produced by the gatherstats program that runs on each host. See the manual page for gatherstats.

2) Doglobals then finds out the names of the hosts available for processing by reading a file called "/usr/tnacct/h_avail". The line in "h_avail" might look like this:

DFHVWXZ

3) For each host represented in h_avail doglobals reads the blkdayread or blkrecentread files and calculates the mean packet length, offered load, transmission and reception rates detected by that particular host.

4) It also totals up the traffic seen by each host and for the entire network for that given day.

5) After each system has been visited, doglobals calculates the global network performance statistics for TOOLSNET, such as: overall mean packet length, total offered load, utilization, hypothetical asymptotic throughput utilization, relative load, perceived efficiency, and average response time.

6) If the "d" switch was selected by the user (usually cron at 6:00 AM) the system and network totals are dumped out to holding files (see OUTPUT FILES, below) for viewing by people and for processing by yet another TAP program, tnet_week.

INPUT FILES

/usr/tnacct/h_avail
/[DFHSVWXZ]/usr/adm/blkdayread
/[DFHSVWXZ]/usr/adm/blkrecentread

OUTPUT FILES

/usr/adm/TAP_TOT_D /usr/adm/TAP_TOT_F
/usr/adm/TAP_TOT_H /usr/adm/TAP_TOT_S
/usr/adm/TAP_TOT_V /usr/adm/TAP_TOT_W
/usr/adm/TAP_TOT_X /usr/adm/TAP_TOT_Z

/usr/adm/TAP_BLK_TOT_D /usr/adm/TAP_BLK_TOT_F
/usr/adm/TAP_BLK_TOT_H /usr/adm/TAP_BLK_TOT_S
/usr/adm/TAP_BLK_TOT_V /usr/adm/TAP_BLK_TOT_W
/usr/adm/TAP_BLK_TOT_X /usr/adm/TAP_BLK_TOT_Z

/usr/adm/TNET_DAY_TOTAL
/usr/adm/TNET_BLK_DAY_TOTAL

The output for the non-BLK files is printed in CG-DBMS format to be used by the shell-scripts taptot and tndt. The *BLK* files are structures (records) consisting of the data structure:

```
typedef struct{
     double rcvd_frames;
     double xmit_frames;
     long excess_collisions;
     long rcvd_collision_fragments;
     long lost_frames;
     long acpt_multicast;
     long rej_multicast;
     long crc_errors;
     long align_errors;
     long collisions;
     long oow_collisions;
     struct {
          double  npkt;
          double  len;
          } len_no[NSYS];
     char stime[NCHARS];
     char sys;
} ac_allstats;
```

Ac_allstats is an accumulation of all the raw statistics managed by TAP.

SEE ALSO

gatherstats(1cg), prr(1cg), sdwatch(1cg), taptot(1cg), tndt(1cg), tnet_week(1cg), totavg(1cg), viewdriv(1cg)

NAME

gatherstats - gather TOOLSNET statisitcs from TOOLSNET driver.

SYNOPSIS

gatherstats

DESCRIPTION

Gatherstats is the statisitics collecting program for the TOOLSNET ANALYSIS PACKAGE (TAP). It runs on every TOOLSNET host after it is invoked by cron. Its main task is to retrieve statistics collected in the TOOLSNET driver and save them in the 4 files mentioned below.

Specifically, gatherstats has three sources of input:

- The stats (if any) held in /usr/adm/tmpfl from viewdriv invocations (see manual for viewdriv)
- Any stats collected by the driver
- A file called, blkrecentread, which contains

  the totals from the driver recorded over the last hour.

Gatherstats has four output files:

- Dayread: the continuously appended file that

  shows the totals collected by the driver for every

  gatherstats invocation for that day. Dayread is in

  CG-DBMS format to afford users the ability to

  use data base tools on it.
- Blkdayread: the binary version of dayread,

  which is used by doglobals for better performance

  by doing block reads, rather than reading in files

  line by line that are formatted for our data base

  package at CG.
- Recentread: same file as blkrecentread, but in CG-DBMS format.
- Blkrecentread: this file is also used for output as explained below.

The statistics held in recentread and dayread are in CG-DBMS format in order to be processed by the prr family of utilities. See the manual for prr. All files produced and read by this program are actually the printing of the following structure:

```
typedef struct{
     long size;     /* Size of allstats */
     char sys; /* System we are looking at. */
     char stime[NCHARS]; /* Sample time. */
     driverstruct drvrsts;    /* Accesses driver stats. */
     }allstats;
```

This structure is composed of the following structures:

```
typedef struct{
     short     zero;
     short     length;
     char phys_adrs[6];
     unsigned short rcvd_frames;
     short     fifo_frames;
     unsigned short xmit_frames;
     short     excess_collisions;
     short     rcvd_collision_fragments;
     short     lost_frames;
     short     acpt_multicast;
     short     rej_multicast;
     short     crc_errors;
     short     align_errors;
     short     collisions;
     short     oow_collisions;
     short     reserved[8];
     char module_id[8];
     char firmware_id[8];
} interlanstruct;

typedef struct{
     short npkt;
     long len; /* Cummulative length of pkts. */
} length_nostruct;

typedef struct{          /* This struct goes to the driver. */
     interlanstruct intrlnsts;
     length_nostruct len_no[NSYS];
} driverstruct;
```

The actual behavior of gatherstats is:

1. Open tmpfl and read it

2. Get statistics from the driver

3. Add 1 to 2

4. Read in blkrecentread

   If there is greater than say, an hour's worth of readings in there (e.g. 13 records, 5 minutes apart)

   Then

   get rid of the first record (line 1)

   append 3 to contents of blkrecentread

   append 3 to contents of recentread

   Else

   just append 3 to contents of blkrecentread

   just append 3 to contents of recentread

5. Write over the entire contents of recentread with the new contents.

6. Append 3 to blkdayread and dayread.

FILES

/usr/adm/dayread /usr/adm/recentread
/usr/adm/blkdayread /usr/adm/blkrecentread
/usr/adm/tmpf1
/sys/h/toolsnet.h
/sys/h/tap.h

SEE ALSO

doglobals(1cg), prr(1cg), sdwatch(1cg), taptot(1cg), tndt(1cg), tnet_week(1cg), totavg(1cg), viewdriv(1cg), cron(8)

NAME

Taptot, taptotsd, taptotin - pretty print contents of TAP_TOT_{D|F|H|S|V|W|X|Z} files.

SYNOPSIS

TAPTOT|taptotsd|taptotin D|F|H|S|V|W|X|Z

DESCRIPTION

This family of utilities prints the contents of certain TOOLSNET ANALYSIS PACKAGE (TAP) tables retained on a central TAP-processing VAX in TOOLSNET.

Taptot pretty prints the entire contents of the /usr/adm/TAP_TOT_? file.

Taptotsd and taptotsd pretty print source and destination data of the file.

Taptotin and taptotin pretty print interlan controller data of the file.

The tables are printed with the following headings:

day: the day that doglobals collected the following stats
rf: the number of received frames detected by the Controller
xf: the number of transmitted frames detected by the Controller
ec: the number of excess collisions detected by the Controller
rc: the number of received collisions detected by the Controller
lf: the number of received collisions detected by the Controller
ce: the number of received collisions detected by the Controller
ae: the number of received collisions detected by the Controller
co: the number of received collisions detected by the Controller
Dn: the number of packets sent to System D
Dl: the number of bytes sent to System D
Fn: the number of packets sent to System F
Fl: the number of bytes sent to System F
Hn: the number of packets sent to System H
Hl: the number of bytes sent to System H
Vn: the number of packets sent to System V
Vl: the number of bytes sent to System V
Wn: the number of packets sent to System W
Wl: the number of bytes sent to System W
Xn: the number of packets sent to System X
Xl: the number of bytes sent to System X
Zn: the number of packets sent to System Z
Zl: the number of bytes sent to System Z
Sn: the number of packets sent to System S
Sl: the number of bytes sent to System S

FILES

/usr/adm/TAP_TOT_D /usr/adm/TAP_TOT_F

/usr/adm/TAP_TOT_H /usr/adm/TAP_TOT_S
/usr/adm/TAP_TOT_V /usr/adm/TAP_TOT_W
/usr/adm/TAP_TOT_X /usr/adm/TAP_TOT_Z

SEE ALSO

doglobals(1cg), gatherstats(1cg), prr(1cg), sdwatch(1cg), tndt(1cg), tnet_week(1cg), totavg(1cg), viewdriv(1cg)

NAME

Tndt, tndtsd, tndtin - pretty print contents of TNET_DAY_TOTAL file.

SYNOPSIS

Tndt|tndtsd|tndtin

DESCRIPTION

This family of utilities prints the contents of a certain TOOLSNET ANALYSIS PACKAGE (TAP) table retained on a central TAP-processing VAX in TOOLSNET.

Taptot pretty prints the entire contents of the /usr/adm/TNET_DAY_TOTAL file.

Taptotsd and taptotsd pretty print source and destination data of the file.

Taptotin and taptotin pretty print interlan controller data of the file.

The tables are printed with the following headings:

day: the day that doglobals collected the following stats
rf: the number of received frames detected by the Controller
xf: the number of transmitted frames detected by the Controller
ec: the number of excess collisions detected by the Controller
rc: the number of received collisions detected by the Controller
lf: the number of received collisions detected by the Controller
ce: the number of received collisions detected by the Controller
ae: the number of received collisions detected by the Controller
co: the number of received collisions detected by the Controller
Dn: the number of packets sent to System D
Dl: the number of bytes sent to System D
Fn: the number of packets sent to System F
Fl: the number of bytes sent to System F
Hn: the number of packets sent to System H
Hl: the number of bytes sent to System H
Vn: the number of packets sent to System V
Vl: the number of bytes sent to System V
Wn: the number of packets sent to System W
Wl: the number of bytes sent to System W
Xn: the number of packets sent to System X
Xl: the number of bytes sent to System X
Zn: the number of packets sent to System Z
Zl: the number of bytes sent to System Z
Sn: the number of packets sent to System S
Sl: the number of bytes sent to System S

FILES

/usr/adm/TNET_DAY_TOTAL

SEE ALSO
doglobals(1cg), gatherstats(1cg), prr(1cg), sdwatch(1cg), taptot(1cg), tnet_week(1cg), totavg(1cg), viewdriv(1cg)

NAME

tnet_week

SYNOPSIS

tnet_week

DESCRIPTION

Tnet_week is part of TOOLSNET ANALYSIS PACKAGE (TAP). Tnet_week sums up the TOOLSNET activity for each system and for TOOLSNET itself for a given work week.

TAP_BLK_TOT_D (binary block i/o files)
TAP_BLK_TOT_F
TAP_BLK_TOT_H
TAP_BLK_TOT_S
TAP_BLK_TOT_V
TAP_BLK_TOT_W
TAP_BLK_TOT_X
TAP_BLK_TOT_Z

TAP_TOT_D (ascii versions)
TAP_TOT_F
TAP_TOT_H
TAP_TOT_S
TAP_TOT_V
TAP_TOT_W
TAP_TOT_X
TAP_TOT_Z

Tnet_week runs at the end of the week and totals up the contents of the *BLK* files and appends the totals to the ascii non-BLK files, in CG-DBMS format.

INPUT FILES

/usr/adm/TAP_BLK_TOT_D /usr/adm/TAP_BLK_TOT_F
/usr/adm/TAP_BLK_TOT_H /usr/adm/TAP_BLK_TOT_S
/usr/adm/TAP_BLK_TOT_V /usr/adm/TAP_BLK_TOT_W
/usr/adm/TAP_BLK_TOT_X /usr/adm/TAP_BLK_TOT_Z
/usr/adm/TNET_BLK_DAY_TOTAL

OUTPUT FILES

/usr/adm/TAP_TOT_D /usr/adm/TAP_TOT_F
/usr/adm/TAP_TOT_H /usr/adm/TAP_TOT_S
/usr/adm/TAP_TOT_V /usr/adm/TAP_TOT_W
/usr/adm/TAP_TOT_X /usr/adm/TAP_TOT_Z
/usr/adm/TNET_DAY_TOTAL

doglobals(1cg), gatherstats(1cg), prr(1cg), sdwatch(1cg), taptot(1cg), tndt(1cg), totavg(1cg), viewdriv(1cg)

NAME

Prr, prrsd, prrin, pdr, pdrsd, pdrin - pretty print dayread or recentread files held in /usr/adm on any TOOLSNET host.

SYNOPSIS

Prr|prrsd|prrin|pdr|pdrsd|pdrin [D|F|H|S|V|W|X|Z]

DESCRIPTION

This family of utilities prints the contents certain TOOLSNET ANALYSIS PACKAGE (TAP) tables retained on each TOOLSNET host. These tables represent the specified host's view of TOOLSNET at the driver level for a certain span of time. The tables are held in /usr/adm/dayread and /usr/adm/recentread (see the manual page for gatherstats for more information about these tables). The tables are printed with the following headings:

time: the time that gatherstats collected the following stats
rf: the number of received frames detected by the Controller
xf: the number of transmitted frames detected by the Controller
ec: the number of excess collisions detected by the Controller
rc: the number of received collisions detected by the Controller
lf: the number of received collisions detected by the Controller
ce: the number of received collisions detected by the Controller
ae: the number of received collisions detected by the Controller
co: the number of received collisions detected by the Controller
Dn: the number of packets sent to System D
Dl: the number of bytes sent to System D
Fn: the number of packets sent to System F
Fl: the number of bytes sent to System F
Hn: the number of packets sent to System H
Hl: the number of bytes sent to System H
Vn: the number of packets sent to System V
Vl: the number of bytes sent to System V
Wn: the number of packets sent to System W
Wl: the number of bytes sent to System W
Xn: the number of packets sent to System X
Xl: the number of bytes sent to System X
Zn: the number of packets sent to System Z
Zl: the number of bytes sent to System Z
Sn: the number of packets sent to System S
Sl: the number of bytes sent to System S

If no host is specified in the argument list, the local host's recentread or dayread file is pretty printed. If a host is specified, that host's recentread or dayread file is pretty printed.

Prr and pdr pretty print the entire contents of the recentread or dayread file.

Prrsd and pdrsd pretty print source and destination data contents of the recentread or dayread file.

Prrin and pdrin pretty print INTERLAN controller data contents of the recentread or dayread file.

YOUR TERMINAL SHOULD HAVE 132 COLUMN CAPABILITY TO FULLY VIEW THESE FILES.

FILES

/[DFHSVWXZ]/usr/adm/dayread
/[DFHSVWXZ]/usr/adm/recentread

SEE ALSO

doglobals(1cg), gatherstats(1cg), sdwatch(1cg), taptot(1cg), tndt(1cg), tnet_week(1cg), totavg(1cg), viewdriv(1cg)

NAME

sdwatch - turn source-destination monitoring on or off.

SYNOPSIS

sdwatch -o|-f

DESCRIPTION

Sdwatch turns the source and destination packet number and packet length counters on or off in the TOOLSNET driver. The source and destination counters provide raw data for TOOLSNET ANALYSIS PACKAGE.

The -o switch turns them on. The -f switch turns them off.

SEE ALSO

doglobals(1cg), gatherstats(1cg), prr(1cg), taptot(1cg), tndt(1cg), tnet_week(1cg), totavg(1cg), viewdriv(1cg)

NAME

TOTAVGD, TOTAVGR - total and average number of packets, and bytes held in any host's dayread or recentread file.

SYNOPSIS

Totavgd|totavgr D|F|H|S|V|W|X|Z

DESCRIPTION

Totavgd prints out the total number of packets and bytes, and their average for packets transmitted by the host(s) named on the command line to all of the other hosts on TOOLSNET, from the beginning of the day.

Totavgr does the same as totavgd, but for the last hour (or whatever length of time is represented in the recentread file). This is especially handy if you suspect that a lot of TOOLSNET activity is slowing the target host down.

SEE ALSO

doglobals(1cg), gatherstats(1cg), prr(1cg), sdwatch(1cg), taptot(1cg), tndt(1cg), tnet_week(1cg), viewdriv(1cg)

NAME

viewdriv [-x]

SYNOPSIS

viewdriv - view TOOLSNET driver information TOOLNET ANALYSIS PACKAGE (TAP).

DESCRIPTION

If no argument is given, viedriv sends the TOOLSNET driver's TAP statistics to both standard output and /usr/adm/tmpfl. The collectors of the statistics are cleared with each invocation of viewdriv. The stats in tmpfl are later processed by another TAP program, gatherstats. If the -x option is given the stats are not saved in tmpfl. This is useful, when it is desireable to clear the collectors.

The statistics collected are shown in the sample output which follows this page.

FILES

/dev/icg - if on the VAXES
/dev/in - if on the PDP-11/70's
/usr/adm/tmpfl

SEE ALSO

doglobals(1cg), gatherstats(1cg), prr(1cg), sdwatch(1cg), taptot(1cg), tndt(1cg), tnet_week(1cg), totavg(1cg)

SAMPLE OUTPUT

```
     Time decimal = 495138615
     Mon Sep  9 14:30:15 1985
          zero = 0(0), length = 62(62), phys adrs = 2 7 1 0 11 3b
          frames received = 0           frames in receive fifo = 0
          frames transmitted =0              excess collisions = 0
          collision fragments received = 0   frames lost 0 times
          multicast frames: accepted = 0          frames rejected = 0
          error frames received: crc = 0          alignment = 0
          collisions = 0                 out of window collisions = 0
          reserved = 0 0 0 0 0 0 0 0
          module_id = N M 1 0 A
          firmware_id = V 0 4 . 0 3

          No. pkts to system D: 0       Total bytes: 0 AVG: 0
          No. pkts to system F: 0       Total bytes: 0  AVG: 0
          No. pkts to system H: 0       Total bytes: 0 AVG: 0
          No. pkts to system V: 0       Total bytes: 0 AVG: 0
          No. pkts to system W: 0       Total bytes: 0 AVG: 0
          No. pkts to system X: 0       Total bytes: 0 AVG: 0
          No. pkts to system Z: 0       Total bytes: 0 AVG: 0
          No. pkts to system S: 0       Total bytes: 0 AVG: 0
```

ript started on Fri Sep 20 17:27:51 1985
doglobals r

ndemental TOOLSNET measurements for system: W
==========
an packet length detected: 201.435394
fered load from this system, Ro: 0.000971
ansmission rate: 6.023636
ception rate: 6.023636

ndemental TOOLSNET measurements for system: V
==========
an packet length detected: 263.447449
fered load from this system, Ro: 0.000249
ansmission rate: 1.182424
ception rate: 1.182424

ndemental TOOLSNET measurements for system: X
==========
an packet length detected: 352.481628
fered load from this system, Ro: 0.000800
ansmission rate: 2.837576
ception rate: 2.838485

ndemental TOOLSNET measurements for system: Z
==========
an packet length detected: 359.667542
fered load from this system, Ro: 0.002264
ansmission rate: 7.869394
ception rate: 7.886061

ndemental TOOLSNET measurements for system: S
==========
an packet length detected: 163.969833
fered load from this system, Ro: 0.000034
ansmission rate: 0.261212
ception rate: 0.261212

ndemental TOOLSNET measurements for system: D
==========
an packet length detected: 107.206650
fered load from this system, Ro: 0.000701
ansmission rate: 8.176970
ception rate: 8.176970

ndemental TOOLSNET measurements for system: F
==========
an packet length detected: 139.551926
fered load from this system, Ro: 0.000461
ansmission rate: 4.125454
ception rate: 4.125454

ndemental TOOLSNET measurements for system: H
==========
an packet length detected: 133.292038
fered load from this system, Ro: 0.000460
ansmission rate: 4.316364
ception rate: 4.303030

LOBAL STATISTICS FOR TOOLSNET==========
ean packet length for network: 215.131561
tilization according to Tannenbaum: 1.032393
tilization according to Stallings: 1.032393
otal offered load, Ro: 0.005941

is the hypothetical number of stations that are ready
o transmit.
is the probability that one station acquires a given slot
n TOOLSNET, for a specific value of Q.
is the mean number of slots devoted to contention
or a certain value of A.
is the mean packet length reported in bytes.
is the amount of time it takes for a signal to travel an
thernet the length of TOOLSNET (200m), and the channel capacity
TOOLSNET (10 Mbps.)
is the percentage that the ethernet channel is utilized for
certain P, Z, and tw.

****

r TOOLSNET with P = 215.132 C = 10000000 bps, tw = 0.000001

e following values are realized:

A	Z	U
1.0000	0.0000	1.0000
0.5000	1.0000	0.9696
0.4444	1.2500	0.9623
0.4219	1.3704	0.9588
0.4096	1.4414	0.9567
0.4019	1.4883	0.9554
0.3966	1.5216	0.9544
0.3927	1.5465	0.9537

pothetical asymptotic throughput effeciency for TOOLSNET
th Q = 1000 stations ready to xmit is: 0.949277

fered load is: 0.005941
lative load is: 0.006258
rceived efficiency is: 0.993742
erage response time is: 0.000022
np

```
tndtin
rint out interlan controller data from TNET_DAY_TOTAL file specified.
at /usr/tnacct/hd2 TNET_DAY_TOTAL | pprint >>  /usr/tnacct/hkey1


ey:
f = received frames, xf = transmitted frames, ec = excess collisions,
c = received collisions, lf = lost frames, ce = checksum error,
e = alignment errors, co = collisions,
)FHVWXZS]n = Number of packets to system [*],
)FHVWXZS]l = Total number of bytes (packet length) to system [*].
ay   rf      xf      ec rc  lf ce   ae   co
---- ------- ------- -- --- -- ---- ---- ---
)n   877438  877486  0  190 0  320  295  124
ie   842663  842661  0  222 0  358  347  112
ed   1160224 1160227 0  147 0  383  368  74
iu   656115  656084  0  84  0  227  219  46
:i   594251  594020  0  95  0  582  577  53
)TAL 4130691 4130478 0  738 0  1870 1806 409
 np
```

```
 taptoin D
aptoin: not found
 taptotin D
rint out interlan controller data from TAP_TOT_D file specified.
at /usr/tnacct/hdl /usr/adm/TAP_TOT_D | pprint >>  /usr/tnacct/hkey1


ey:
f = received frames, xf = transmitted frames, ec = excess collisions,
: = received collisions, lf = lost frames, ce = checksum error,
e = alignment errors, co = collisions,
)FHVWXZS]n = Number of packets to system [*],
)FHVWXZS]l = Total number of bytes (packet length) to system [*].
ime  rf     xf     ec rc  lf ce  ae  co
---- ------ ------ -- --- -- --- --- --
)n   88955  88881  0  24  0  160 160 11
ıe   84965  84878  0  30  0  201 201 7
ed   154965 154856 0  18  0  232 232 13
ıu   130429 130375 0  12  0  157 157 8
:i   197504 197443 0  16  0  143 143 19
)TAL 656818 656433 0  100 0  893 893 58
 np
```

```
taptotin Z
rint out interlan controller data from TAP_TOT_Z file specified.
at /usr/tnacct/hd2 /usr/adm/TAP_TOT_Z | pprint >>  /usr/tnacct/hkey1


ey:
 = received frames, xf = transmitted frames, ec = excess collisions,
 = received collisions, lf = lost frames, ce = checksum error,
 = alignment errors, co = collisions,
)FHVWXZS]n = Number of packets to system [*],
)FHVWXZS]l = Total number of bytes (packet length) to system [*].
ıy   rf     xf      ec rc  lf ce  ae  co
---- ------ ------ -- --- -- --- --- --
)n   107371 107326 0  28  0  98  98  21
ıe   106869 106814 0  35  0  121 121 10
:d   126954 126905 0  14  0  123 123 15
ıu   135546 135541 0  14  0  47  47  9
:i   145415 145189 0  20  0  421 421 14
)TAL 622155 621775 0  111 0  810 810 69
np
```

totavgr F

otal transmits, packets and bytes transmitted by System F
==========

ange of times for these measurements: Sep 20 16:01:01 to Sep 20 16:56:01

---

otal number of packets transmitted by system F:
3530
****
otal number of packets, bytes, and averages transmitted to D by F:
okt_sum: 74 nbyte_sum: 31173 average: 421.000000
****
otal number of packets, bytes, and averages transmitted to F by F:

****
otal number of packets, bytes, and averages transmitted to H by F:
okt_sum: 2864 nbyte_sum: 292072 average: 101.000000
****
otal number of packets, bytes, and averages transmitted to V by F:
okt_sum: 836 nbyte_sum: 90505 average: 108.000000
****
otal number of packets, bytes, and averages transmitted to W by F:
okt_sum: 50 nbyte_sum: 5052 average: 101.000000
****
otal number of packets, bytes, and averages transmitted to X by F:
okt_sum: 3212 nbyte_sum: 655322 average: 204.000000
****
otal number of packets, bytes, and averages transmitted to Z by F:
okt_sum: 6494 nbyte_sum: 808632 average: 124.000000

```
: ls /H/unix
'H/unix
 viewdriv
'ime decimal = 496100336
'ri Sep 20 17:38:56 1985
      zero = 0(0), length = 62(62), phys adrs = 2 7 1 0 11 3b
      frames received = 8                      frames in receive fifo = 0
      frames transmitted =8                    excess collisions = 0
      collision fragments received = 0         frames lost 0 times
      multicast frames: accepted = 0           frames rejected = 0
      error frames received: crc = 0           alignment = 0
      collisions = 0                           out of window collisions = 0
      reserved = 0 0 0 0 0 0 0 0
      module_id = N M 1 0 A
      firmware_id = V 0 4 . 0 3

      No. pkts from system D: 0       Total bytes: 0  AVG: 0
      No. pkts from system F: 0       Total bytes: 0  AVG: 0
      No. pkts from system H: 8       Total bytes: 724        AVG: 90
      No. pkts from system V: 0       Total bytes: 0  AVG: 0
      No. pkts from system W: 0       Total bytes: 0  AVG: 0
      No. pkts from system X: 0       Total bytes: 0  AVG: 0
      No. pkts from system Z: 0       Total bytes: 0  AVG: 0
      No. pkts from system S: 0       Total bytes: 0  AVG: 0
 np
```

```
# pr   s

<<<<<< TOOLSNET TRAFFIC FROM SYSTEM S TO OTHER SYSTEMS >>>>>>


Key:
rf = received frames, xf = transmitted frames, ec = excess collisions.
rc = received collisions, lf = lost frames, ce = checksum error,
ae = alignment errors, co = collisions,
[DFHVWXZS]n = Number of packets to system [*],
[DFHVWXZS]l = Total number of bytes (packet length) to system [*].
```

time	rf	xf	ec	rc	lf	ce	ae	co	Dn	Dl	Fn	Fl	Hn	Hl	Vn	Vl	Wn	Wl	Xn	Xl	Zn	Zl	Sn	Sl
Sep 20 16:01:00	264	264	0	0	0	0	0	0	124	43512	0	0	0	0	0	0	0	0	0	0	0	0	140	14296
Sep 20 16:06:01	12	12	0	2	0	0	0	0	12	1124	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sep 20 16:11:01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sep 20 16:16:01	148	148	0	0	0	0	0	0	0	0	0	0	0	0	0	0	148	25344	0	0	0	0	0	0
Sep 20 16:21:01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sep 20 16:26:01	228	228	0	0	0	0	0	0	0	0	0	0	0	0	0	0	228	34044	0	0	0	0	0	0
Sep 20 16:31:01	32	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32	3274	0	0	0	0	0	0
Sep 20 16:36:01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sep 20 16:41:01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sep 20 16:46:01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sep 20 16:51:02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Sep 20 16:56:01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
# np
```

<<<<<< TOOLSNET TRAFFIC FROM SYSTEM V TO OTHER SYSTEMS >>>>>>

Values for Systems (Dn, Dl, Hn, etc.,) are in the 1000's.

Key:
rf = received frames, xf = transmitted frames, ec = excess collisions,
rc = received collisions, lf = lost frames, ce = checksum error,
ae = alignment errors, co = collisions,
[DFHVWXZS]n = Number of packets to system [*],
[DFHVWXZS]l = Total number of bytes (packet length) to system [*].

day	rf	xf	ec	rc	lf	ce	ae	co	Dn	Dl	Fn	Fl	Hn	Hl	Vn	Vl	Wn	Wl	Xn	Xl	Zn	Zl
Mon	130615	130635	0	19	0	25	0	12	32.2	12005.1	7.7	2390.7	4.6	1423.6	0.0	0.0	27.0	3310.8	4.0	989.3	53.0	74
Tue	47853	47849	0	25	0	7	0	12	9.6	1187.1	7.4	1925.6	1.3	172.5	0.0	0.0	3.0	548.1	2.3	409.1	23.9	57
Wed	61989	61994	0	22	0	6	0	10	21.7	3417.2	10.3	2598.8	1.7	224.1	0.0	0.0	1.9	257.6	0.9	132.0	24.6	30
Thu	46643	46649	0	14	0	4	0	3	15.8	2885.1	6.6	2286.5	0.9	166.3	0.0	0.0	0.5	52.9	0.3	40.5	22.4	38
Fri	8947	8947	0	1	0	0	0	1	3.5	994.9	1.0	320.1	0.7	172.9	0.0	0.0	1.3	193.6	0.2	21.0	2.3	57
TOTAL	296047	296074	0	81	0	42	0	38	82.9	20489.4	33.0	9521.7	9.2	2159.4	0.0	0.0	33.6	4363.0	7.8	1591.9	126.2	20

np

```
# tnotso
Print out source and destination data from TNET_DAY_TOTAL file specified.

<<<<<<<<<<<<<<<< Traffic to Systems is in 1000's >>>>>>>>>>>>>>>>>>>>

PUT YOUR TERMINAL IN 132 COLUMNS TO READ THIS:

Key:

[OFHVWXZS]n = Number of packets to system [*],
[OFHVWXZS]1 = Total number of bytes (packet length) to system [*].
```

day	On	D1	Fn	F1	Hn	H1	Vn	V1	Wn	W1	Xn	X1	Zn	Z1	Sn
Mon	89.3	20808.6	132.8	25773.7	110.8	14310.8	129.8	29220.7	40.5	5210.1	43.3	6476.7	107.7	17757.0	222.6
Tue	85.1	15278.1	188.9	30746.8	157.1	24221.8	47.5	7147.5	37.5	6926.3	59.7	8716.7	106.5	18632.7	159.9
Wed	220.0	44392.2	217.6	39565.6	181.2	38836.7	127.0	26520.5	78.6	15164.0	133.3	7962.7	191.9	32406.0	10.2
Thu	115.6	22644.1	133.5	42737.8	91.4	15639.8	45.4	10173.5	28.4	7655.4	95.9	12253.9	130.6	16339.5	15.1
Fri	77.2	19103.4	102.2	30961.6	84.1	18941.9	30.3	4077.6	112.0	13698.6	76.8	10319.6	106.6	13582.3	4.6
TOTAL	587.4	122226.4	775.0	169785.5	624.7	111950.9	379.9	77139.8	297.0	48654.5	409.1	45729.6	643.3	98717.6	412.3

```
#
script done on Fri Sep 20 17:43:04 1985
```

APPENDIX C

Glossary of Terms

Bit transfer rate - The number of bits transferred per unit time, usually expressed in bits per second (bps).

Bus - See channel.

Channel - That part of a communications system that connects a message source to a message sink. A path for electrical transmission between two or more points. Also called a circuit, facility, line, link, or path.

Channel capacity - A term which expresses the maximum baud rate that can be handled by the channel.

Collision - When one station's signal interferes with another's on the same transmission channel.

Computer network - An interconnection of assemblies of computer systems, terminals and communications facilities.

Contention - A condition on a communications channel when two or more stations tgry to transmit at the same time.

Control procedure - The means used to control the orderly communication of information between stations on a data link. Also call: line discipline. See also: protocol

Controller - A device which regulates network traffic from the host to the channel and conversely.

Cron - A UNIX program which reads a file called crontab. Crontab is a list of programs to be executed at a specific time.

CSMA - Carrier Sense Multiple Access. In a CSMA LAN, such as Ethernet, a node listens to the channel before transmitting, and if the channel is idle begins transmitting. However, there is a chance that two or more nodes may here an idle channel and transmit at the same time, or one of them may transmit slightly before the other transmits, because it has not heard the other's signal due to propagation delay. Either scenario results in a collision.

Data communication - The interchange of data messages from one point to another over communications channels.

Data transmission - The sending of data from one place for reception elsewhere.

Error control - An arrangement that detects the presence of errors. In some systems, refinements are added that will correct the detected errors, either by operations on the received data or by transmission from the source. Examples: Check sum errors, alignment errors, and collisions.

Header - The control information prefixed in a message text, e.g., source or destination information, priority, or message type.

Local area network - A computer network which is usually contained within 1 kilometer, and is usually owned and managed by and individual organization.

Packet - A group of bits including data and control elements which is transmitted as a composite whole (package). The data and control elements and possibly error control information are arranged in a specified format (see [TAN 83]).

Port - A general term for a source or sink in a computer network. An example would be a terminal which generates or receives character data at 9600 bits per second.

Propagation delay - The amount of time it takes for a transmitted signal to move from its origin to another point on a channel.

Protocol - A formal set of conventions governing the format and relative timing of message exchange between two communicating prosesses. See 'control procedure'.

Response time - The elapsed time between the generation of the last character of a message at a terminal delay, network delay, and service node delay.

Virtual Circuit - A connection between a source and a sink in a network that may be realized by different circuit configurations during transmissions of of a message. See [TAN 81] for a far more detailed explanation.

APPENDIX D

Annotated Bibliography and Reading List

[ALM_79]

Guy T. Almes and Edward Lazowska; "The behavior of Ethernet-like networks"; Proceedings 7th Symposium on OS Principles Dec. 79.

This is an excellent paper that explains various performance measurements for CSMA/CD bus networks. A good introduction to Ethernet-like networks is given.

[AME_82]

P. D. Amer; "A measurement center for the NBS local area computer network"; IEEE Trans on Computers; Vol. C-31, No. 8, August 1982.

Very informative article, gives both background information and design concerns for network management systems. Very informative and well worth reading.

[BAI_83]

R.M. Bailey, R.C. Soucy; "Performance and availability measurement of the IBM Information Network"; IBM Systems Jour-

nal; Vol 22. no 4, 1983.

This paper describes measurement techniques to track performance of user-service attributes of the IBM Information Network.

[CHL_84]

I. Chlamtac, R. Jain; "A methodology for building a simulation model for efficient design and us performance analysis of local area networks"; Simulation; February 1984.

[CLA_78]

David D. Clark, Kenneth T. Pogran, and David P. Reed; "An Introduction to Local Area Networks"; Proceedings of the IEEE; Vol._66, NO. 11, November 1978.

Classic Paper. Widely discusses major aspects of local area networks: topologies, protocols, network interconnecting, network control structures, contention control strategies, reliability considerations, hardware-host interactions, distributed operating systems, distributed computing and many other topics. Paper is easy to read and well worth the time.

[DAV_83]

D. B. Davis; "Network management and control: different meanings to different vendors"; Mini-Micro Systems; February 1983.

Different modes of network monitoring are discussed. This paper shows clearly that the data communications industry took the lead over academia in developing network monitors.

[DON_79]

James E. Donnelley; "Components of a Network Operating System"; Computer Networks; Vol. 3, Dec. 1979

A message passing network operating system is described. A message passing approach to network transparency is contrasted with a system call approach at the operating system level.

[EBI_83]

Yoshihiko Ebihara, Katsua Ikeda, Tomoo Nakamura, Maichihiro Ishizaka, Makaoto Shinzawa,and Kazuhiko Nakayama; "Gamma-Net: A Local Computer Network Coupled by a High Speed Optical Fiber Ring Bus - System Concept and Structure"; Computer Networks; 1983

Thoroughly describes a 32 Mbps Fiber Ring Bus Network. Interesting features: complex IMPS; 4 OSI layers described: Data link control layer, Network control layer, Function control layer, and Application layer; loosely coupled distributed operating system functions, including network monitoring system.

[ELL_73]

C. Ellingson and R. Kulpinski; "Dissemination of system time"; IEEE Trans. Commun.; Vol COM-21, May 1973

Discusses classic problem of synchronizing timing devices. This article was quoted in [AME 82].

[ELO_80]

Honey S. Elovitz and Constance L. Heitmeyer; "What Is a Computer Network"; Reprinted with permission from: NTC 1974 Record, pp. 1007-11014. In: Computer Networks: Text and references for a tutorial, 1980. IEEE Computer Society.

Discusses aspects of transparency afforded by network operating systems by contrasting networks with operating systems against ones with out them. Good introductory article.

[EXC_83]

"The Excelan Nutcracker(TM) LAN System Analyzer/Simulator for Ethernet, Preliminary Product Description;" Copyright 1983, by Excelan Inc.

This is a comprehensive description of this state-of-the art tool.

[HEY_83]

D. P. Heyman; "Data-transport performance analysis of Fastnet"; The Bell System Technical Journal; Vol._62, no._8,

Oct. 1983.

Discusses methods for calculating mean packet delay on a token passing network.

[INT_83]

"How to Use the INTERLAN NS4240, XEROX ITP NETWORK SOFTWARE (ITP/UNIX)"; Copyright 1983 by INTERLAN, Inc.

On pages 122-148, NETMAN is described.

[KLE_64]

L. Kleinrock; "COMMUNICATION NETS Stochastic Message Flow and Delay"; USA: McGraw-Hill, 1964.

Doctoral thesis, discussing wide variety of topics in processes theory. The chapter in this book that is relevant to my thesis is the discussion of mean packet delay.

[KLE_70]

L. Kleinrock; "Analytic and simulation methods in computer network design"; Spring Joint Computer Conference, AFIPS 1970

Discusses several ways of calculating average packet delay, drawn from queuing theory. It also discusses ways of estimating costs on the Arpanet.

[MAT_82]

E. Matsukane; " Network administration and control system for a broadband local area communications network"; Compcon 82 Fall Proceedings, Washington, DC, September 20-23, 1982

Discusses a network management system that not only collects and processes network behavior statistics. The design is briefly described for the AMDAX CableNet(tm). Very little background information is given. It is a good article nonetheless.

[MET_76]

Robert M. Metcalfe and David R. Boggs; "Ethernet: Distributed Packet Switching for Local Computer Networks"; Communications of ACM; Vol. 19, July 1976.

Classic paper outlining aspects of Ethernet as it was introduced into the scientific community. Since this time Ethernet has become a trend setter in Local Area Network topology design. It will be interesting to see if the buss structure will have any ramifications upon the design of network operating systems. Paper is definitely worth reading.

[MER_82]

T. Mercer; "Performance measurement--on-line data base for telecom professionals"; Telecommunications; Vol. 16, no. 12, Nov. 1982

This article outlines practical approaches for monitoring network performance.

[MUR_84]

D. N. Murray and P. H. Enslow, Jr.; "An Experimental Study of the Performance of a Local Area Network"; IEEE Communications Magazine; Vol. 22, no. 11, Nov. 1984

This paper is an excellent study of many of the delay features encountered on a local area network.

[NUT_82]

G. J. Nutt and D. L. Bayer; "Performance of CSMA/CD networks under combined voice and data loads"; IEEE Trans. on Com; Vol. 30, no. 1, Jan. 1982

This paper describes a simulation study which shows that voice data does not significantly degrade performance of an Ethernet.

[OGL_81]

J. Ogle, K. Damanda, "The evolution of automatic monitoring within a data communications network"; Telecommunications; Vol. 15, no. 3, Mar. 1981

This paper discusses both the practical and economical reasons behind the development on network performance monitors. It discusses attributes of these systems which make them valuable to have.

[SHA_83]

R. R. Shatzer; "Local Area Network Upper Level

Standardization"; SPIE Vol. 418 Picture Archiving & Communication Systems (PACS II) for Medical Applications, 1983

A fairly detailed overview of OSI architecture. Shatzer has a utopian idea that standardization will one day lead to universal compatibility of all networks. A good paper nonetheless.

[SHO_80]

J. F. Shoch and J. A. Hupp; "Performance of an Ethernet Local Network--A Preliminary Report"; COMPCON; Spring 1980

A good paper for reporting performance of an Ethernet Lan. Demonstrates use of network monitor as a statistical gathering device and experimental testbed.

[STA-84]

William Stallings; "Local Network Performance"; IEEE Transactions on Communications; Vol. 22, Feb. 84

This paper stresses the importance of measuring network utilization as a performance measurement in local area networks. It shows how to calculate utilization for CSMA/CD, token ring, and token bus. The paper contains comparative studies between these three protocols. It is definitely worth reading.

[TAN_81]

Andrew S. Tannenbaum Computer Networks, Prentice Hall, Inc.,

Englewood Cliffs, N. J. 07632, 1981.

APPENDIX E

TAP CODE

The first sub-appendix contains include files which are global to TAP and the TAP code inserts into TOOLSNET code. The second sub-appendix contains files which belong only to the gatherstats family of modules. The third sub-appendix contains files that are part of the doglobals family of modules. The fourth sub-appendix contains files that are shared by doglobals and tnet_week. The fifth sub-appendix is tnet_week. The sixth sub-appendix contains programs which either stand alone or are used with some of the shell scripts mentioned in Appendix B.

E.1 Include files and TAP inserts

```
* "%W% %G%" */
include <stdio.h>
include "/sys/h/toolsnet.h"
ndef NSYS
efine NSYS 8
efine C              10000000     /* bits per second Capacity */
efine tw             .000001      /* (5 microsecs/1km) = (1 microsecs/.2 km) */
                                  /* TOOLSNET LENGTH IS 200 m. */
efine BYTE           8            /* 8 BITS/BYTE. */
efine NCHARS         26           /* Number of chars in date */
efine dbg            0
efine dbgl           0
efine dbgms(x) dbg ? printf("%s\n",x) : dbg == 0
efine SYSDOWN        -1
efine MAXSHORT       32767
efine MAXLONG        2147483647
efine OBSERVED_MPL   150          /* Use me to tune data when integer overflows */
                                  /* are encountered.     */
efine FUDGE 200                   /* FUDGE stats if overflow */
efine MAXRECS          12         /* Max number of records in recentread files */
efine MAXPACKET        1518       /* MAX BYTES IN AN ENET PACKET. */
LE *fp;

pedef struct{
      long size;                  /* Size of allstats */
      char sys;                   /* System we are looking at. */
      char stime[NCHARS];
      driverstruct drvrsts;
      }allstats;

tern allstats st;
tern char *tm;

ruct ratestats{                   /*  This structure will be used to hold the
                                  **  occurences/sec and the mean packet len
                                  */
      char    sys;                /* System we are looking at. */
      float   rcvd_persec;
      float   xmit_persec;
      float   mpl;                /* Mean packet length. */
      float   Ro;                 /* Average offered load. */



pedef struct{
      double rcvd_frames;
      double xmit_frames;
      long    excess_collisions;
      long    rcvd_collision_fragments;
      long    lost_frames;
      long    acpt_multicast;
      long    rej_multicast;
      long    crc_errors;
      long    align_errors;
      long    collisions;
      long    oow_collisions;
```

```
      struct {
              double  npkt;
              double  len;                    /* Cummulative length of pkts. */
              } len_no[NSYS];
      char stime[NCHARS];
      char sys;
ac_allstats;

xtern struct ratestats ratesarray[];
```

```
define NSYS 10
define HARD_RESET 1             /* For explicitly reseting the driver from */
define SOFT_RESET 2             /* an ioctl call to it. */
define ON_COUNTERS 3            /* For turning on source and destination cnts*/
define OFF_COUNTERS 4

ypedef struct{
      short   zero;
      short   length;
      char    phys_adrs[6];
      unsigned short rcvd_frames;
      short   fifo_frames;
      unsigned short  xmit_frames;
      short   excess_collisions;
      short   rcvd_collision_fragments;
      short   lost_frames;
      short   acpt_multicast;
      short   rej_multicast;
      short   crc_errors;
      short   align_errors;
      short   collisions;
      short   oow_collisions;
      short   reserved[8];
      char    module_id[8];
      char    firmware_id[8];
interlanstruct;

ypedef struct{
      short npkt;                       /* Number of packets received. */
      float len;                        /* Cummulative length of pkts. */
length_nostruct;

ypedef struct{                          /* This struct goes to the driver. */
      interlanstruct intrlnsts;
      length_nostruct len_no[NSYS];
driverstruct;


' This stuff is for sdpair traffic. */

lefine D_SYS 86         /* WAS 56 last hex digits of host id */
lefine F_SYS 79
lefine H_SYS 159        /* -97  */
lefine V_SYS 11
lefine W_SYS 59
lefine X_SYS 245        /* -11 */
lefine Z_SYS 19
lefine S_SYS 14

lefine SH 100.00                /* converts shorts to fp to save room */
```

TAP INSERTS INTO TOOLSNET CODE.
==========

To the reader of this thesis: the first page shows the call to the source and destination tracking counters of TAP on the VAXES. The context of this call is the subroutine "xmit_pkt", through which all data packets pass.

The next page shows "sdpair" which collects the packet length and pkt number counters per host.

The next pages show the "read" routine of the driver where TAP retrieves both the controller's statistics and these packet statistics

The next page shows the "ioctl" routine of the driver.

```
nit_pkt(release, pp)            /* returns 0 if failed, 1 if passed */
truct pkt_hd *pp;

      register struct epkt *ep;
      register int i;
      register int retries = 0;
      struct rmt_link *cur;

if(count_pkts)                  /* Get destination accounting. */
    sdpair(pp);
uidacct(pp,u.u_ruid);

      u.u_error = 0;
      do{
              for( i = retries * RETRY_INTERVAL; i; i--){
                      sleep((caddr_t)&lbolt, NET_PRIORITY);
              }

              icg_submit(0, pp);        /* will pend until completed */

              if( ! (pp->ph_flags & PH_RETRY) ){
                  if( pp->ph_flags & PH_ERROR ){
                      .
                      .
                      .
```

```
* Collect source and destination traffic.  Won't include acks and naks.   */
* on the VAXES, because call is in higher layer.  On 11/70's does include */
* acks and nacks because call is in the driver itself.  The difference    */
* is due to our room problem on the 11/70's -- much of TOOLSNET [illegible]  */
* resides in a kernel overlay and will generate an overlay fault when     */
* the collecting is done in a higer layer there.                          */

* These are defined in the TOOLSNET DRIVER, icg.c (VAX), in.c (PDP). */

* The values are passed back to the user in the read() routine of the
* driver.
/

xtern short dno; extern long dlen;
xtern short fno; extern long flen;
xtern short hno; extern long hlen;
xtern short vno; extern long vlen;
xtern short wno; extern long wlen;
xtern short xno; extern long xlen;
xtern short zno; extern long zlen;
xtern short sno; extern long slen;


dpair(ip)
:ruct pkt_hd *ip;

      register struct epkt *ep;
      unsigned int sysid = 0;
      ep = (struct epkt *) map_in_pkt(ip);



      sysid = ep->n_un.ne_r_source[5] & 0xFF;

      switch (sysid)
      {
          case D_SYS: dlen +=  ep->ne_length; ++dno; break;
          case F_SYS: flen +=  ep->ne_length; ++fno; break;
          case H_SYS: hlen +=  ep->ne_length; ++hno; break;
          case V_SYS: vlen +=  ep->ne_length; ++vno; break;
          case W_SYS: wlen +=  ep->ne_length; ++wno; break;
          case Z_SYS: zlen +=  ep->ne_length; ++zno; break;
          case S_SYS: slen +=  ep->ne_length; ++sno; break;
          case X_SYS: xlen +=  ep->ne_length; ++xno; break;
          default: printf("ICG.C: Unknown source : %d\n", sysid & 0xFF);
       }
/*sdpair*/
```

```
/* In TAP the routines getstats calls icg_read for the program gatherstats.  */
/* The program viewdriv, also calls icg_read.                                */

#include "/sys/h/toolsnet.h"

icg_read(dev)
dev_t dev;

        register struct pkt_hd *pp;

        register  driverstruct *sp;
        register interlanstruct *ns;
        driverstruct drvst;
        length_nostruct *l_no;

        static int d1 = 1, d2 = 1;
        register struct a {
                int     fdes;
                char    *cbuf;
                unsigned count;
        } *uap = (struct a *)u.u_ap;


        sp = &drvst;

        l_no = sp->len_no;
        for(;;){
                pp = get_pkt();          /* get a place to put statistics */
                if( pp != NULL ){        /* quit if a packet is available */
                        break;
                }
                sleep(&lbolt, NET_PRIORITY);    /* otherwise, wait for one */
        }
        icg_command(0, ICG_DO_STATS, pp->ph_xmem << 14, pp->ph_dev_addr, 66);

        ns  = (interlanstruct *) map_in_pkt(pp);   /* put controller */
        sp->intrlnsts = *ns;
                                                        /* counters in sp   */
        l_no->npkt = dno;
        l_no->len = dlen;
        l_no++;


        l_no->npkt = fno;
        l_no->len= flen;
        l_no++;


        l_no->npkt = hno;
        l_no->len = hlen;
        l_no++;

        l_no->npkt = vno;
        l_no->len= vlen;
        l_no++;

        l_no->npkt = wno;
        l_no->len= wlen;
        l_no++;

        l_no->npkt = xno;
        l_no->len= xlen;
        l_no++;
```

```
l_no->len = zlen;
l_no++;

l_no->npkt = sno;
(l_no)->len = slen;

dlen=flen=hlen=vlen=wlen=xlen=zlen=slen= 0;
dno=fno=hno=vno=wno=xno=zno=sno= 0;

bcopy((caddr_t)sp, (caddr_t)uap->cbuf, uap->count);

map_out_pkt(pp);
rels_pkt(pp);
```

```
* This is the ioctl routine of the TOOLSNET driver.  Just showing the
* code for the sdwatch call.
/

cg_ioctl(dev, cmd, addr, flag)
ev_t dev;
addr_t addr;

  int status;
  int unit;
  uid_struct *uidp;
  int fd;
  int i;

  register struct device *dvp = icg_addr[minor(dev)];

  unit = minor(dev);            /* Just to be sure. */

  switch (cmd)
  {
     case ON_COUNTERS: count_pkts = TRUE;      /* for sdwatch */
            break;

     case OFF_COUNTERS: count_pkts = FALSE;
            break;

     OTHER CODE FOLLOWS, NOT APPLICABLE TO THESIS.

  } /* end switch */

return(status);
```

E.2 Gatherstats code

```
/* static char filename[] = " %W% %G% "; */

/*************************************************************************
```

Gatherstats will be invoked automatically on a daily basis by the operating system. It will run every 5 minutes, on each host. It's main purpose is to invoke the driver to retrieve statistics from the controller, and to append these statistics, to two files.

Specifically, gatherstats has three sources of input:

- The stats (if any) held in tmpfl from viewdriv invocations
- Any stats collected by the driver
- A file called, blkrecentread, which contains the totals from the controller over the last time interval, deemed as recent history for network behavior, for example, the last half hour's worth of network performance statistics

Gatherstats has four output files:

- Dayread: the continuously appended file that shows the totals collected by the driver for every invocation for that working day. Dayread is in CG-DBMS format to afford users the ability to use data base tools on it.
- Blkdayread: the binary version of dayread, which is used by doglobals for better performance by doing block reads, rather than reading in files line by line that are formatted for our data base

at CG.

- Recentread: The totals collected by the driver for the last hour in DBMS format.

- Blkrecentread: the binary version of recentread.

The actual behavior of this program is to:

1. Get statistics from the driver
2. Add 1 to 2
3. Read in blkrecentread

    If there is greater than say, an hour's worth of readings in there

        Then

            get rid of the first line of totals

            append 3 to contents of blkrecentread

            append 3 to contents of recentread

        Else

            just append 3 to contents of blkrecentread

            just append 3 to contents of recentread
4. Write over the entire contents of recentread with the new contents.
5. Append 2 to blkdayread and dayread.

```
*************************************************************************

#include "tap.h"
allstats st;

char tmpfl[] = "/usr/adm/tmpfl";
char recentread[] = "/usr/adm/recentread";
char blkrecentread[] = "/usr/adm/blkrecentread";
char dayread[] = "/usr/adm/dayread";
char blkdayread[] = "/usr/adm/blkdayread";

/* The purpose of gatherstats is to get the statistics from the driver,
** combine them with statistics saved in tempfl, and to update the files:
** recentread, blkrecentread, dayread, and blkdayread.
**
```

```
main()
{
FILE *fp;

    getstats();          /* get stats from driver */

    combinestats();      /* update the stats that were just read in*/

    updaterecentread();
    updatedayread();

} /* main */
```

```
/* Combinestats takes the stats held in st and adds TO them the older
** statistics that are saved in tmpfl (if it exists).  This way, network
** statistics generated by the program viewdriv are not lost.  Rather
** they add to the accumulation of the day's network performance statistics.
*/
combinestats()

    /* Uses global variable: st defined in tap.h */

    int j;                                  /* Value holder */
    allstats temp,*accum;

    if (dbg) printf("combinestats\n");
    fp = fopen(tmpfl,"r");      /*  Open up userstats for a read. */
    if (fp == NULL)
    {
        return;

    }

    dbgms("stat before add");
    if(dbg)dbgwrite(&st);
    dbgms("\n\n");
    while ((j= fread(&temp,sizeof(temp),1,fp)) != 0  ) /* read to EOF */
    {
        if (dbg) printf("j = %d\n",j);
        dbgms("HERE IS TEMP");
        if(dbg) dbgwrite(&temp);
        addstats(&st,&temp);                    /* Add the stats together. */
                                                /* See file addstats.c */
    } /* endwhile */

    fclose(fp);

    fp = fopen(tmpfl,"w");                      /* Erase contents, don't want
                                                ** any of them old stat hangin
                                                ** around
                                                */

    fclose(fp);
    dbgms("stat after add");
    if(dbg) dbgwrite(&st);

    dbgms("\n\n");
  /*end combinestats*/
```

```
/* This procedure appends the final version of the driver's statistics
** (i.e. they have been through combinestats and addstats) to the two
** files that keep them for further processing by the rest of NAP:
** dayread and blkdayread.
*/

updatedayread()
{
    FILE *fpp;
    int i;                          /* loop control */
    allstats temp;       /* Use for formatted write. */
    char time[9];                   /* just print out the time */
    int j, k= 0;                    /* loop control */

    dbgms("updatedayread");


    fpp = fopen(dayread,"a");

     if (fpp == NULL){
         printf("Can't open %s\n", dayread); /*Something is really wrong here */
         exit(1);
     } /*endif */

    formwrite(fpp,&st);
    fclose(fpp);

    fpp = fopen(blkdayread,"a");

    st.size = sizeof(st);                   /* For 11 - VAX compatibility. */

    fwrite(&st,sizeof(st),1,fpp);

    fclose(fpp);

} /* end udatedayread*/
```

```
/* Updaterecentread appends the same record that updatedayread() appends to
** two files "recentread" and "blkrecentread".  It is  also responsible for
** removing the oldest record in these files.
*/

updaterecentread()
{
#define NSLOTS 13                /* 1 hour + 1 slot at 5 min/slot */
FILE *recptr;
FILE *brecptr;                   /* block reads */
allstats sbuf[NSLOTS], *s;
int atend;
int i;
int ret;                         /* Gets EOF from formread() */


    dbgms("updaterecentread");

    brecptr = fopen(blkrecentread,"r+");


    if (brecptr != NULL)                /* It is there */
    {
          int j;

          dbgms("Before fscanf");

          while ((j= fread(&sbuf[i],sizeof(st),1,brecptr)) != 0  )
          {
              i++;
              if (i == NSLOTS)
                 break;
          }
          rewind(brecptr);

    }


    if (dbg) printf("NSLOTS = %d i = %d\n",NSLOTS,i);

    recptr = fopen(recentread,"w");     /* Write over contents of recentread */

        if(recptr == NULL)
        {
          printf("can't open %s\n", recentread);
          exit(0);
        }

    if (i == NSLOTS)                    /* Append st to buffer */
    {

        sbuf[NSLOTS-1] = st;


        dbgms("Before formwrite");
```

```
                                        /* Recentread is "full", so we get
                                        ** rid of the first record by
                                        ** starting to print at the 2nd record
                                        */

    for(i = 1; i < NSLOTS; i++)
    {
       formwrite(recptr,&sbuf[i]);
       if (dbg) printf("i = %d\n");
    }
    dbgms("made past formwrite");

    if(brecptr == NULL)                 /*Virgin blkrecentread file. */
        brecptr = fopen(blkrecentread,"w");
    if (brecptr == NULL)
    {
       puts("can't open %s\n", blkrecentread);
       exit(0);
    }



    for(i = 1 ; i < NSLOTS;i++)
    {
       sbuf[i].size = sizeof(sbuf[i]);      /* for pdp/vax compat */
       fwrite(&sbuf[i],sizeof(sbuf[i]),1,brecptr);
    }

}/*endif*/
else
{

  if (dbg)printf("Here is i: %d \n");
  sbuf[i] = st;
  atend = i;

  dbgms("\nHERE IS SBUF[I]");

  if(dbg) dbgwrite(&sbuf[i]);
  dbgms("Before 2nd brecptrintf");

  if (dbg) printf("i=atend: %d\n",atend);

  i = 0;

  while(i <= atend)
  {
      if (dbg)printf("Here is i again: %d\n",i);
      formwrite(recptr,&sbuf[i]);
      i++;
  }
    dbgms("BEFORE BLOCK RECENTREAD");

    if (brecptr == NULL)
        brecptr = fopen(blkrecentread,"w");
    if (brecptr == NULL)
```

```
        {
           printf("can't open %s\n", blkrecentread);
           exit(0);
        }

        for(i = 0; i <= atend;i++)
        {
           sbuf[i].size = sizeof(sbuf[i]);              /* for pdp/vax compat */
           fwrite(&sbuf[i],sizeof(sbuf[i]),1,brecptr);
        }

    } /*endelse*/
```

```
/* This procedure is called by gatherstats to read both the counters
** from the driver and the registers that the driver reads off of the
** Interlan Controller.
*/
#include "tap.h"
getstats()
{
        register fd;

        if (dbg) printf("getstats\n");
        fd = open("/dev/icg", 0);
        if( fd < 0 ){
                printf("can't open\n");
                exit(1);
        }

        read(fd, &st.drvrsts, sizeof(st.drvrsts));
        close(fd);

        gettime();
        if (dbg) dbgwrite(&st);


/* This procedure fills the global varible tm with the current time */

gettime()
{
    long thetime,time();
    char *ctime();
    thetime = time(0);

    sprintf(st.stime,"%s",ctime(&thetime));
}
```

```
/* static char filename[] = " %W% %G% "; */
#include "tap.h"

formwrite(fpp,stat)
FILE *fpp;
allstats *stat;
{
     int i;

     char TIME[20];
     char *s;
     char *p = &TIME[0];

     for (s = &stat->stime[4]; s <= &stat->stime[4] + 14; s++)
         *p++ = *s;
     *p = NULL;


     fprintf(fpp,"%s;%d;%d;%d;%d;%d;%d;%d;%d;", /* d;d;d;d; */


                 TIME,
                 stat->drvrsts.intrlnsts.rcvd_frames,
                 /*
                 stat->drvrsts.intrlnsts.fifo_frames,
                 */
                 stat->drvrsts.intrlnsts.xmit_frames,
                 stat->drvrsts.intrlnsts.excess_collisions,
                 stat->drvrsts.intrlnsts.rcvd_collision_fragments,
                 stat->drvrsts.intrlnsts.lost_frames,
                 stat->drvrsts.intrlnsts.crc_errors,
                 stat->drvrsts.intrlnsts.align_errors,
                 stat->drvrsts.intrlnsts.collisions
                 /*
                 stat->drvrsts.intrlnsts.oow_collisions,
                 stat->drvrsts.intrlnsts.acpt_multicast,
                 stat->drvrsts.intrlnsts.rej_multicast
                 */);

#ifdef VAX
    for (i = 0; i < NSYS; i++)
        fprintf(fpp,"%d;%ld;",stat->drvrsts.len_no[i].npkt * 2,
                 stat->drvrsts.len_no[i].len * 2 );
    fputc('\n',fpp);
#else
    for (i = 0; i < NSYS; i++)
        fprintf(fpp,"%d;%ld;",stat->drvrsts.len_no[i].npkt ,
                 stat->drvrsts.len_no[i].len );
    fputc('\n',fpp);
#endif
}
```

```
/* static char filename[] = " %W% %G% "; */

/* Use dbgwrite when you need to look at an allstats structure for debugging
** purposes.
*/
#include "tap.h"


dbgwrite(stat)
allstats *stat;

     int i;
     printf("%s;%d;%d;%d;%d;%d;%d;%d;%d;%d;%d;%d;%d;",
                stat->stime,
                stat->drvrsts.intrlnsts.rcvd_frames,
                stat->drvrsts.intrlnsts.fifo_frames,
                stat->drvrsts.intrlnsts.xmit_frames,
                stat->drvrsts.intrlnsts.excess_collisions,
                stat->drvrsts.intrlnsts.rcvd_collision_fragments,
                stat->drvrsts.intrlnsts.lost_frames,
                stat->drvrsts.intrlnsts.crc_errors,
                stat->drvrsts.intrlnsts.align_errors,
                stat->drvrsts.intrlnsts.collisions,
                stat->drvrsts.intrlnsts.oow_collisions,
                stat->drvrsts.intrlnsts.acpt_multicast,
                stat->drvrsts.intrlnsts.rej_multicast);

    for (i = 0; i < NSYS; i++)
        printf("%d\t%f\n",stat->drvrsts.len_no[i].npkt * 2, stat->drvrsts.len_no[

    dbgms("leaving formwrite");
```

```
#include "tap.h"
/* %W% %G% */

/* The purpose of ac_addstats is to accumulate the the TOOLSNET statistics
** accumulated in two ac_allstat structures.  For example, to add up all of
** the activity on one system (tmp) to an accumulator that represents all of
** the TOOLSNET activity for one day.
**
** Overflow of the driver's registers is vary rare because gatherstats runs
** about every 5 minutes.  However, it must be detected, even at such a
** late stage in TAP.  Compensating steps are taken to make its impact as
** small as posible on the big picture.
*/

static char overflow = FALSE;

ac_ac_addstats(ac,new)
ac_allstats *ac;
ac_allstats *new;
{
    char *p,*q;


    ac->rcvd_frames += new->rcvd_frames;
    ac->xmit_frames += new->xmit_frames;
    ac->excess_collisions += new->excess_collisions;
    ac->rcvd_collision_fragments +=new->rcvd_collision_fragments;
    ac->lost_frames += new->lost_frames;
    ac->crc_errors += new->crc_errors;
    ac->align_errors += new->align_errors;
    ac->collisions += new->collisions;
    ac->oow_collisions += new->oow_collisions;
    ac->acpt_multicast  += new->acpt_multicast ;
    ac->rej_multicast += new->rej_multicast;
    ac_ac_a_plen(ac,new);
    ac_ac_a_pno(ac,new);
    if(overflow)
    {
        adjust_counters(ac);
    }
    overflow = FALSE;
  /* addstats */
```

```
ac_ac_a_plen(ac,new)
ac_allstats *ac;
ac_allstats *new;
{
    int i;

    for (i = 0; i < NSYS; i++)
    {
        if (
            (new->len_no[i].len < 0)             /* Overflow. */
                    ||
            (new->len_no[i].len < new->len_no[i].npkt)
           )
        {
                overflow = TRUE;

                new->len_no[i].len = MAXSHORT * FUDGE;
                new->len_no[i].npkt = MAXSHORT;
        }
        ac->len_no[i].len += new->len_no[i].len;
    }/*endfor*/
}
```

```
ac_ac_a_pno(ac,new)
ac_allstats *ac;
ac_allstats *new;
{
    int i;

    for (i = 0; i < NSYS; i++)
    {
        if (
                (new->len_no[i].npkt < 0) /* Overflow. */
                        ||
                (new->len_no[i].npkt > new->xmit_frames)
           )

        {
                overflow = TRUE; /* Fudge to best posible overflow (minimum) */
                new->len_no[i].len = MAXSHORT * FUDGE;
                new->len_no[i].npkt = MAXSHORT;
        }
        ac->len_no[i].npkt += new->len_no[i].npkt;
    }/* endfor */
}
```

```
/* %W% %G% */

/* The purpose of ac_addstats is to accumulate the the TOOLSNET statistics
** compiled on every system.
**
** Because it is handling raw statistics, overflow of the driver's counters
** must be protected against, by reducing the impact of it on the big
** picture.  Overflow is very rare, and has been reduced to almost nothing
** since reducing the sample time of the driver by gatherstats to once every
** 5 minutes.
*/

#include "tap.h"

char overflow = FALSE;
ac_addstats(ac,tmp)
ac_allstats *ac;
allstats *tmp;
{
    char *p,*q;


    ac->rcvd_frames += (long)tmp->drvrsts.intrlnsts.rcvd_frames;
    ac->xmit_frames += (long)tmp->drvrsts.intrlnsts.xmit_frames;
    ac->excess_collisions += (long)tmp->drvrsts.intrlnsts.excess_collisions;
    ac->rcvd_collision_fragments +=(long)tmp->drvrsts.intrlnsts.rcvd_collision_fra
    ac->lost_frames += (long)tmp->drvrsts.intrlnsts.lost_frames;
    ac->crc_errors += (long)tmp->drvrsts.intrlnsts.crc_errors;
    ac->align_errors += (long)tmp->drvrsts.intrlnsts.align_errors;
    ac->collisions += (long)tmp->drvrsts.intrlnsts.collisions;
    ac->oow_collisions += (long)tmp->drvrsts.intrlnsts.oow_collisions;
    ac->acpt_multicast  += (long)tmp->drvrsts.intrlnsts.acpt_multicast ;
    ac->rej_multicast += (long)tmp->drvrsts.intrlnsts.rej_multicast;
    ac_a_plen(ac,tmp);
    ac_a_pno(ac,tmp);
    if(overflow)
    {
        adjust_counters(ac);
    }
    overflow = FALSE;
} /* addstats */

ac_a_plen(ac,tmp)
ac_allstats *ac;
allstats *tmp;
{
    int i;

    for (i = 0; i < NSYS; i++)
    {
        if (tmp->drvrsts.len_no[i].npkt == 0)
            continue;

        if (
            (tmp->drvrsts.len_no[i].len < 0)            /* Overflow. */
                        ||
```

```
            (tmp->drvrsts.len_no[i].len < tmp->drvrsts.len_no[i].npkt)
           )
        {
                overflow = TRUE;

                tmp->drvrsts.len_no[i].len = MAXSHORT * FUDGE;
                tmp->drvrsts.len_no[i].npkt = MAXSHORT;
        }
        if(ac->sys < 'I')       /* 11/70's contain acks + data sent out */
            ac->len_no[i].len += tmp->drvrsts.len_no[i].len;
        else
            ac->len_no[i].len += tmp->drvrsts.len_no[i].len * 2; /* Just data */
    }/*endfor*/
}



ac_a_pno(ac,tmp)
ac_allstats *ac;
allstats *tmp;
{
    int i;

    for (i = 0; i < NSYS; i++)
    {
        if (
                (tmp->drvrsts.len_no[i].npkt < 0) /* Overflow. */
                        ||
                (tmp->drvrsts.len_no[i].npkt >
                        tmp->drvrsts.intrlnsts.xmit_frames)
           )

        {
                overflow = TRUE; /* Fudge to best posible overflow (minimum) */
                tmp->drvrsts.len_no[i].len = MAXSHORT * FUDGE;
                tmp->drvrsts.len_no[i].npkt = MAXSHORT;
        }
        if (ac->sys < 'I')
            ac->len_no[i].npkt += tmp->drvrsts.len_no[i].npkt;
        else
            ac->len_no[i].npkt += tmp->drvrsts.len_no[i].npkt * 2;
    }/* endfor */
}
```

```
/* static char filename[] = " %W% %G% "; */
#include "tap.h"
#define SCALE    1000

ac_formwrite(fpp,stat)
FILE *fpp;
ac_allstats *stat;
{
     int i;

     char TIME[20];
     char *s;
     char *p = &TIME[0];

     for (s = &stat->stime[0]; s <= &stat->stime[0] + 3; s++)
         *p++ = *s;
     *p = NULL;



     fprintf(fpp,"%s;%.0f;%.0f;%d;%d;%d;%d;%d;%d;",
                TIME,
                stat->rcvd_frames,
                stat->xmit_frames,
                stat->excess_collisions,
                stat->rcvd_collision_fragments,
                stat->lost_frames,
                stat->crc_errors,
                stat->align_errors,
                stat->collisions
                /*
                stat->oow_collisions,
                stat->acpt_multicast,
                stat->rej_multicast
                */);

/* Only do this on VAXEN */

    for (i = 0; i < NSYS; i++)
        fprintf(fpp,"%.1f;%.1f;",stat->len_no[i].npkt / SCALE,
        stat->len_no[i].len  / SCALE );
    fprintf(fpp,"%c",'\n');
}
```

```
#include "tap.h"
/* @(#)addstats.c       1.6 9/14/85 */

/* The purpose of addstats is to add the totals gleaned from the driver's
** counters to an accumulator, needed for further processing.
**
** Overflow is not protected against here.  These statistics are the raw
** stats viewed by the TOOLS GROUP.  If there is overflow going on we want
** to know about it.
*/

addstats(ac,tmp)
allstats *ac,*tmp;
{
    char *p,*q;


    ac->drvrsts.intrlnsts.rcvd_frames += tmp->drvrsts.intrlnsts.rcvd_frames;
    ac->drvrsts.intrlnsts.fifo_frames += tmp->drvrsts.intrlnsts.fifo_frames;
    ac->drvrsts.intrlnsts.xmit_frames += tmp->drvrsts.intrlnsts.xmit_frames;
    ac->drvrsts.intrlnsts.excess_collisions += tmp->drvrsts.intrlnsts.excess_colli
    ac->drvrsts.intrlnsts.rcvd_collision_fragments +=tmp->drvrsts.intrlnsts.rcvd_c
    ac->drvrsts.intrlnsts.lost_frames += tmp->drvrsts.intrlnsts.lost_frames;
    ac->drvrsts.intrlnsts.crc_errors += tmp->drvrsts.intrlnsts.crc_errors;
    ac->drvrsts.intrlnsts.align_errors += tmp->drvrsts.intrlnsts.align_errors;
    ac->drvrsts.intrlnsts.collisions += tmp->drvrsts.intrlnsts.collisions;
    ac->drvrsts.intrlnsts.oow_collisions += tmp->drvrsts.intrlnsts.oow_collisions;
    ac->drvrsts.intrlnsts.acpt_multicast  += tmp->drvrsts.intrlnsts.acpt_multicast
    ac->drvrsts.intrlnsts.rej_multicast += tmp->drvrsts.intrlnsts.rej_multicast;
    a_plen(ac,tmp);
    a_pno(ac,tmp);
} /* addstats */

a_plen(ac,tmp)
allstats *ac, *tmp;
{
    int i;

    for (i = 0; i < NSYS; i++)
    {
        if (tmp->drvrsts.len_no[i].npkt == 0)   /* Length == 0  w/npkt == 0 */
        {
            continue;
        }
        if (tmp->drvrsts.len_no[i].len < 0) /* Overflow. */
        {
                tmp->drvrsts.len_no[i].len = MAXSHORT * FUDGE;
                tmp->drvrsts.len_no[i].npkt = MAXSHORT;
        }
        ac->drvrsts.len_no[i].len += tmp->drvrsts.len_no[i].len;
    }
}



a_pno(ac,tmp)
```

```
allstats *ac, *tmp;
{
    int i;

    for (i = 0; i < NSYS; i++)
    {
        if (tmp->drvrsts.len_no[i].npkt < 0) /* Overflow. */
        {
            tmp->drvrsts.len_no[i].npkt = tmp->drvrsts.len_no[i].len
                                          / OBSERVED_MPL;
        }
        ac->drvrsts.len_no[i].npkt += tmp->drvrsts.len_no[i].npkt;
    }
}

double received_mpl(ac)
allstats *ac;
{
    int i;
    float acnpkt= 0.0;
    double aclen = 0.0;

    for (i = 0; i < NSYS; i++)
        if (ac->drvrsts.len_no[i].len < 0) /* Overflow. */
        {
                ac->drvrsts.len_no[i].len = MAXSHORT * FUDGE;
                ac->drvrsts.len_no[i].npkt = MAXSHORT;
        }
        else
           aclen += ac->drvrsts.len_no[i].len;

    for (i = 0; i < NSYS; i++)
        if(ac->drvrsts.len_no[i].npkt < 0)
        {
                ac->drvrsts.len_no[i].len = MAXSHORT * FUDGE;
                ac->drvrsts.len_no[i].npkt = MAXSHORT;
        }
        else
           acnpkt += ac->drvrsts.len_no[i].npkt;


    if(acnpkt > 0)
        return(aclen / acnpkt);
     return(0);
}
```

```
/*"%W% %G%"*/

/*      The bulk of the calculations found in this file are taken from
**      [ALM 79].  These calculations are described in the Utilization
**      section of /usr/cgdoc/tap.doc.  A thesis by Daniel Sorrentino.
*/


#include "tap.h"

#define N_ASYMP 1000                    /* Number of stations for asymptotic
                                        ** throughput efficiency.
                                        */


#define relative_load(ro,asyU) (ro/asyU) /*Offered load / asymptotic throughput
                                          ** efficiency.
                                          */

#define perceived_efficiency(rl) (1 - rl)

#define average_response_time(mpl,c,pe) (mpl/c/pe)

int Q;                  /* Number of stations that desire a given slot, at
                        ** any one moment.
                        */
float q;
double A[NSYS];         /* Probability that one station acquires a given
                        ** slot.
                        */

double Z[NSYS];         /* Mean number of slots devoted to contention.  */

double Ualm[NSYS];       /* Utilization of TOOLSNET.  */

float asyU;             /* Asymptotic throughput efficiency. */

float RL;               /* Relative Load. */

float AR;               /* Average Response Time. */

float PE;               /* Perceived Efficiency */


   /* Calculate hypothetical states of TOOLSNET determined by changing
   ** values of "Q", which is the number of stations that desire to transmit.
   ** For each state, calculate the probabality that one station will
   ** acquire the cable, the mean number of slots devoted to contention,
   ** and the amount that the network is being utilized.
   */


#include <math.h>
```

```
int nsys;                   /* Actual number of systems read in. */

almes_79(Ro,mpl,rarr)
float Ro;                   /* Offered load */
float mpl;                  /* Mean packet length.*/
struct ratestats *rarr;
{
    double pow();

    nsys = calc_nsys(rarr);     /* Get number of systems that were up and
                                ** included in these global calculations.
                                */


    for (Q = q =1;Q <= nsys; Q++, q = Q)
    {
        if (Q == 1)
            A[Q] = (double)1.0; /* See definitions in [MET 76]. */
        else
        {
            A[Q]  = pow((double)(1 - 1/q),(double)(q-1));
        }



        Z[Q]  = (double) ((Q == 1) ?  0 :  (1 -A[Q]) / A[Q]) ;

        Ualm[Q]  = (mpl * BYTE/C) /((mpl * BYTE/C) + (5.4 * tw * Z[Q]) );
    }/*endfor*/

    printast(mpl);

    /* Assign parameters needed for asymptotic throughput efficiency. */
    /* The factor that makes asymptotic throughput efficiency greater than */
    /* 1/e is the mean packet length, being greater than a minimal packet. */

    Q = N_ASYMP;
                                                /* A = .37, 1/e */
    Z[0] = 1.703;                               /* (1 - A) / A */
    asyU = (mpl * BYTE/C) /((mpl * BYTE/C) + (5.4 * tw * Z[0])) ;

    printf("\nHypothetical asymptotic throughput effeciency for TOOLSNET\n");
    printf("with Q = %d stations ready to xmit is: %f\n\n", N_ASYMP, asyU);

    printf("Offered load is: %f\n", Ro);
    printf("Relative load is: %f\n", RL = relative_load(Ro,asyU));

    printf("Perceived efficiency is: %f\n",PE = perceived_efficiency(RL));

    printf("Average response time is: %f\n",AR=average_response_time(mpl,C,PE));

}
```

```
/* Any sytem that has been read in will have the rarr->sys file set to
** the system ID, otherwise it will be NULL.  This function determines the
** number of systems involved.
*/
calc_nsys(rarr)
struct ratestats *rarr;
{
    int cnt = 0;
    while(rarr->sys)
    {
        ++rarr;
        cnt++;
    }
    return(cnt);
}

printast(P)
float P;                /* Mean packet length. */
{
    int i;

    printf("Q is the hypothetical number of stations that are ready\n");
    printf("to transmit.\n");
    printf("A is the probability that one station acquires a given slot\n");
    printf("on TOOLSNET, for a specific value of Q.\n");
    printf("Z is the mean number of slots devoted to contention\n");
    printf("for a certain value of A.\n");
    printf("P is the mean packet length reported in bytes.\n");
    printf("tw is the amount of time it takes for a signal to travel an");
    printf("Ethernet the length of TOOLSNET \(200m\), and the channel capacity");
    printf("of TOOLSNET \(10 Mbps.\)");
    printf("U is the percentage that the ethernet channel is utilized for\n");
    printf("a certain P, Z, and tw\n\n");
    printf("************************************************\n");
    printf("For TOOLSNET with P = %.3f C = %d bps, tw = %f \n\n", P, C, tw);

    printf("The following values are realized:\n\n");

    printf("Q\tA\tZ\tU\n");


    for (i = 1; i <= nsys ; i++)
       printf("%d\t%.4f\t%.4f\t%.4f\n",i,A[i],Z[i],Ualm[i]);
}
```

```
/* " @(#)doglobals.c    1.6 9/18/85 "  */

/*************************************************************************
```

The purpose of doglobals is to calculate global performance statistics for TOOLSNET. The statistics it calculates are mentioned below.

It behaves in the following way:

1) Doglobals is executed by the user with a "d" or an "r" switch. The "d" switch tells doglobals to obtain the raw measurements its raw measurements from the blkdayread files from each host. The "r" switch tells doglobals to use the blkrecentread files from each host. Remember "blkdayread" and "blkrecentread" are produced by the gatherstats program that runs on each host.

2) Doglobals then finds out the names of the hosts availble for processing by reading a file called "h_avail". The line in "h_avail" might look like this:

        DFHVWXZ

3) For each host represented in h_avail doglobals reads the blkdayread or blkrecentread files and calculates the mean packet length, offered load, transmission and reception rates detected by that particular host. See the procedure, getrates() in the code below.

4) It also totals up the traffic seen by each host and for the entire network for that given day.

5) After each system has been visited, doglobals calculates the global network performance statistics for TOOLSNET, such as:

        overall mean packet length,
        total offered load,
        utilization according to [TAN 81] (eq. 6),
        utilization according to [MUR 84] (eq. 7),
        utilization according to [ALM 79] (eq. 4),
        hypothetical asymptotic throughput efficiency (eq. 5),
        relative load (eq. 8),
        perceived efficiency (eq. 9),
                and
        average response time (eq. 10).

   See the procedure, calcglobals().

6) If the "d" switch was selected by the user (usually cron at 6:00 AM) the system and network totals are dumped out to holding files for viewing by people and for processing by yet another program, tnet_week (see tnet_week.c).

   See the procedure pub_totals for further documentation concerning these holding files.

****

```
#include "tap.h"
#include <math.h>


#define  UTAN(ml) ( (ml * BYTE / C) / ( (ml * BYTE/C)-(5.4 * tw) )  )
#define  USTA(ml) ( (l) / ((l) - ((5.4 * tw) / (ml * BYTE / C)) ) )
#define  CTRL_L '\014'
#define  MAXSYS 40

int rflag = 0;

char tmpfl[] = "/usr/adm/tmpfl";
char recentread[] = "/usr/adm/recentread";
char blkrecentread[] = "/usr/adm/blkrecentread";
char dayread[] = "/usr/adm/dayread";
char blkdayread[] = "/usr/adm/blkdayread";
char h_avail[] =  "/usr/tnacct/h_avail";


char TAP_TOT_SYS[] = "/usr/adm/TAP_TOT_@";
char TAP_BLK_TOT_SYS[] = "/usr/adm/TAP_BLK_TOT_@";
char TNET_DAY_TOTAL [] = "/usr/adm/TNET_DAY_TOTAL";
char TNET_DAY_BLK_TOTAL[] = "/usr/adm/TNET_DAY_BLK_TOTAL";
char hosts[MAXSYS];

struct ratestats rates= {0,0,0,0,0};

char file[50] ;
char targfile[20];

allstats st;
c_allstats accum;                /* An accumulator for driver statistics. *
c_allstats accum_array[MAXSYS]; /* For holding daily totals for each host.
c_allstats *ac_arr;              /* For pointing to accum_array. */

truct ratestats ratesarray[MAXSYS];
```

```
main(argc,argv)
int argc;
char **argv; {

    ac_allstats *ac;            /* Running total of driver stats */

    ac_allstats ac_day; /*Totals up traffic for TOOLSNET for a day. */

    struct ratestats *rts;
    struct ratestats *rarr;
    char *p, sys;
    char *thefile;
    FILE *fp;

    ac =  &accum;
    rts = &rates;

    zeroaccum(&st);
    zeroaccum(ac);
    zeroaccum(&ac_day);

    rarr = &ratesarray[0];
    ac_arr = &accum_array[0];

    fp = fopen(h_avail,"r");
    if (fp == NULL)
    {
        printf("No h_avail file in /usr/tnacct . Bye. \n");
        exit();
    }
    fgets(hosts,80,fp);

    if (argc == 1)
    {
       puts("Either dayread (d) or recentread (r) needed on command line-bye");
       exit(1);
    }
    else if (argv[1][0] == 'd')
       strcpy(targfile,blkdayread);
    else if (argv[1][0] == 'r')
       strcpy(targfile,blkrecentread);
    else
    {
       printf("Bad format-bye.");
       exit(1);
    }
    if(argv[1][0] == 'r')       /* defend against inconsistent files */
        rflag = TRUE;           /* Fix later.  Works w/getrates fread */

    for(p = hosts; *p != '\n'; ++p)  /* Replace by user interface*/
    {                                /** according to design in document.*/

        sys = *p;

         sprintf(file,"/%c%s",sys,targfile);
```

```
        /* GOOD CODE: JUST NOT APPROVED YET */

        /* strcpy(file,targfile);       GET RID OF WHEN YOU HAVE FOUND A HOME *
        ac->sys  = sys;
        rarr->sys = sys;
        getrates(file,rts,ac);          /* When I speak of <rates> I mean
                                        ** the totals gleaned from the driver
                                        ** divided by the total time to
                                        ** get the number of events, that each
                                        ** total represents, per second.
                                        */
        if(rts->xmit_persec != -1)
        {
            *rarr = *rts;
            ++rarr;

            *ac_arr = *ac;                      /* Save for each host. */
            ac_ac_addstats(&ac_day,ac); /* Total up for network for this day.*/
            ++ac_arr;

            zeroaccum(ac);      /* Reset the accumulator structure */
        }

    }/*endfor*/

    calcglobals(ratesarray);

    if(argv[1][0] == 'd')
    {
       ac_arr = &accum_array[0];        /* Reset address. */
       pub_totals(ac_arr,&ac_day);
    }

/*main*/
```

```
/* For a given system and either file: blkrecentread or blkdayread
** getrates totals up each field, calculates the duration of time held in that
** file (represented by the last time - the first time) and determines how
** many events/sec occured.
*/
char first[NCHARS] = NULL, last[NCHARS] = NULL, nlast[NCHARS];
getrates(file,rts,ac)
char *file;
struct ratestats *rts;
ac_allstats *ac;
{

    long gettimedif();
    float received_mpl();
    int i;
    int nrecs = 0;
    allstats new;
    static double dur;
    static int dur_calculated = FALSE;


    fp = fopen(file,"r");
    if (fp == NULL)                         /* System could be down. */
    {
       rts->xmit_persec = -1;
       return;
    }

    if(ac->sys < 'I')    /* Treating data from PDP-11'S DIFFERENTLY: D F H */
    {
        do
        {
            pdp_getallstats(&new,fp);
            if(new.stime[0] == NULL)
               break;              /* reached eof. */
            ac_addstats(ac,&new);
            if(dur_calculated == FALSE)
            {
                if(first[0] == NULL)
                    strcpy(first,new.stime);
                strcpy(nlast,new.stime);
            }
        }while(1);
    }/* endif*/
    else
        while(( i = fread(&new,sizeof(new),1,fp)) != 0) /*Total up the fields.*/
        {
           if(rflag && dur_calculated == FALSE)
           {
                if(nrecs == MAXRECS) /* Inconsistent file bug-fix */
                {
                    break;
                }
            }
           nrecs++;
```

```
            ac_addstats(ac,&new);
            if(dur_calculated == FALSE)
            {
                if(first[0]  == NULL)
                    strcpy(first,new.stime);
                strcpy(nlast,new.stime);                /* Save last time. */
            }


        } /*end while */

    if(dur_calculated == FALSE)
    {
        strcpy(last,nlast);
        strcpy(ac->stime,last);          /* Just save the date part. */
    }                                                    .

        rts->sys = ac->sys;
        rts->mpl = received_mpl(ac);             /* Get mean packt length. */

        if(dur_calculated == FALSE)
            dur = (double)gettimedif(first,last);
        dur_calculated = TRUE;

        if(dur < 0)              /* Something goofed up in time. */
        {

            printf("Negative time. Exiting.\n");
            exit();
        }


         if(rts->mpl)

            get_offered_ld(rts,ac,dur);         /* Actually calculate the
                                                ** rates
                                               */
         else
            rts->Ro = 0;

         *first = *last = NULL;
/*end getrates */
```

```
get_offered_ld(rts,ac,duration)
struct ratestats *rts;
ac_allstats *ac;
double duration;
{

        dbgms("get_offered_ld");


        duration =  (duration == 0) ? 1 :  duration; /* Don't divide by 0! */


        rts->rcvd_persec = ac->rcvd_frames/duration;
        rts->xmit_persec = ac->xmit_frames/duration;
        rts->Ro = rts->xmit_persec / (C / (rts->mpl * 8)); /* 8 bits/byte */

}
```

```
/* This procedure calculates the global performance parameters:
** Utilization [TAN 81]
** Utilization [STA 84]
*/
calcglobals(rarr)
struct ratestats *rarr;
{

float   mpl,                    /* Mean packet length */
        Utan,                   /* Efficiency/Utilization according to Tan */
        Usta,                   /* Efficiency/Utilization by Stalings */
        Ro,
        Ro_total(),             /* Offered load of network. */
        overallmpl();           /* Mean packet length for whole network */


    dbgms("calcglobals");

    mpl = overallmpl(rarr);
    Ro = Ro_total(rarr);
    Utan = UTAN(mpl);

    Usta = USTA(mpl);

    pubrarr(rarr);
    publishglobs(mpl,Utan,Usta,Ro);
    almes_79(Ro,mpl,rarr);
}
```

```
pubrarr(rarr)
struct ratestats *rarr;
{
     struct ratestats *r;

     for ( r = rarr; r->sys ; r++)
     {
        printf("\nFundemental TOOLSNET measurements for system: %c\n", r->sys);
        printf("================================================\n");
        printf("Mean packet length detected: %f\n", r->mpl);
        printf("Offered load from this system,  Ro: %f\n", r->Ro);
        printf("Transmission rate: %f\n", r->xmit_persec);
        printf("Reception rate: %f\n", r->rcvd_persec);
        putchar('\n');
     }
}
```

```
publishglobs(mpl,Utan,Usta,Ro)
float mpl, Utan, Usta, Ro;
{

      dbgms("publishglobs");
      putchar(CTRL_L);
      printf("\nGLOBAL STATISTICS FOR TOOLSNET");
      printf("==============================\n");
      printf("Mean packet length for network: %f\n",mpl);
      printf("Utilization according to Tannenbaum: %f\n",Utan);
      printf("Utilization according to Stallings: %f\n", Usta);
      printf("Total offered load, Ro: %f\n",Ro);
      putchar(CTRL_L);
}
```

```
/* Pub_totals() creates or appends to 2 files per network host.  Each file
** stores the total statistics gleaned from the dayread files on each host
** for a given work day.  The 2 files per host are named TAP_TOT_@ and
** TAP_TOT_BLK_@, where the @ is replaced by a host name: D | F | H | V | W |
** X | Z | S.  These files show their view of TOOLSNET for a whole week.
**
** In addition, 2 other files are created or appended: TNET_DAY_TOTAL and
** TNET_BLK_DAY_TOTAL.  These files contain the total traffic stats on TOOLSNET
** for a whole day.
**
** All of the above files will be used by another program: tnet_week.c that
** will process them and report TOOLSNET's behavior for a week.
**
** This procedure is called only when this program is called with the -d option.
*/

pub_totals(ac_arr,ac_day)
ac_allstats *ac_arr;            /* For daily pt of view of TOOLSNET for sys @ */
ac_allstats *ac_day;            /* For daily pt of view of TOOLSNET. */
{

    FILE *fp, *bfp;
    char *txt, *blk;
    char emsg[80];
    char *err;
    int i;

    txt = TAP_TOT_SYS;
    blk = TAP_BLK_TOT_SYS;
    err = emsg;

    while(ac_arr->sys)
    {
        txt[strlen(txt) - 1] = ac_arr->sys;     /*Replace @ with host id. */
        blk[strlen(blk) - 1] = ac_arr->sys;     /*   ditto                */

        fp = fopen(txt,"a");
        if(fp == NULL)
        {
            printf("Can't open %s\n",txt);
            perror(err);
            exit();
        }
        ac_formwrite(fp,ac_arr);        /* Dump stats out for a system */

        bfp = fopen(blk,"a");
        if(bfp == NULL)
        {
            printf("Can't open %s\n",blk);
            perror(err);
            exit();
        }
        fwrite(ac_arr,sizeof(*ac_arr),1,bfp);
```

```
        ac_arr++;
        fclose(fp);
        fclose(bfp);

    } /* endwhile*/
    ac_arr--;                                /* Save a day for ac_day */
    for(i = 0; i < 10; i++)
        ac_day->stime[i] = ac_arr->stime[i];
    for(i = i; i < NCHARS; i++)
        ac_day->stime[i] = NULL;
    fp = fopen(TNET_DAY_TOTAL,"a");

    ac_formwrite(fp,ac_day);

    bfp = fopen(TNET_DAY_BLK_TOTAL, "a");
    fwrite(ac_day,sizeof(*ac_day),1,bfp);

}/*pub_totals*/
```

```
#include "tap.h"
#include <ctype.h>
#define HOUR 3600
#define MIN 60
char tic[3];

long gettimedif(fst,lst)
char *fst, *lst;
{
    long ft;
    long lt;
    tic[2] = NULL;      /* For atoi conversion. */

    ft = gethour(fst) * HOUR;
    ft += getmin(fst) * MIN;
    ft += getsec(fst);

    lt = gethour(lst) * HOUR;
    lt += getmin(lst) * MIN;
    lt += getsec(lst) ;

    return(lt - ft);
}

gethour(tictoc)         /* Convert hours into seconds. */
char *tictoc;                           /* Provide most flexible way to do it.*/

    char *p;
    long x;

    for(p = tictoc; *p != ':' ; p++);    /* Parse e.g. Jun 30 20:15:59 */

    p = p - 2;
    tic[0] = *p++;
    tic[1] = *p;

    x = atoi(tic);
    return(x);



getmin(tictoc)                          /* Convert hours into seconds. */
char *tictoc;                           /* Provide most flexible way to do it.*/

    char *p;

    for(p = tictoc; *p != ':' ; p++);    /* Parse e.g. Jun 30 20:15:59 */

    tic[0] = *(++p);
    tic[1] = *(++p);

    return(atoi(tic));
```

```
getsec(tictoc)                          /* Convert seconds. */
char *tictoc;                           /* Provide most flexible way to do it.*/
{
    char *p;

    for(p = tictoc; *p != ':' ; p++);    /* Parse e.g. Jun 30 20:15:59 */

    for(p = p+1; *p != ':' ; p++);    /* Parse e.g. Jun 30 20:15:59 */

    tic[0] = *(++p);
    tic[1] = *(++p);

    return(atoi(tic));
}
```

```
#include "tap.h"
/* This function averages the mean packet length from each host to
** calculate the mean of the means.
*/

float overallmpl(rarr)
struct ratestats *rarr;          /* rarr contains the rates of each host */
{
     float acmpl = 0.0;
     struct ratestats *r;
     int i, nsys;

     for (r = rarr, nsys = 0; r->sys ; r++, nsys++)
     {
         if (r->mpl > 0)
           acmpl += r->mpl;
     }
     if(nsys > 0)
        return(acmpl/(nsys));
     else
     {
        printf("overallmpl: No systems. Bye.\n");
        exit();
     }

}
```

```
/* Received_mpl calculates the mean packet length received by a given host. */
#include "tap.h"

double received_mpl(ac)
ac_allstats *ac;
{
    double mpl = 0.0;
    int i;
    double acnpkt= 0.0;
    double aclen = 0.0;

    for (i = 0; i < NSYS; i++)
        if (ac->len_no[i].len < 0) /* Overflow. */
        {
                ac->len_no[i].len = MAXSHORT * FUDGE;
                ac->len_no[i].npkt = MAXSHORT;
        }
        else
           aclen += ac->len_no[i].len;

    for (i = 0; i < NSYS; i++)
        if(ac->len_no[i].npkt < 0)
        {
                ac->len_no[i].len = MAXSHORT * FUDGE;
                ac->len_no[i].npkt = MAXSHORT;
        }
        else
           acnpkt += ac->len_no[i].npkt;


    if(acnpkt > 0)
    {
        mpl = aclen / acnpkt;
        if(mpl > MAXPACKET)              /* GOT A POSITIVE SNEAKY OVERFLOW. */
           return(OBSERVED_MPL);  /* Yet another defense against overflow.*/
        else
           return(mpl);
    }
    else
        return(0);
```

```
/* Ro_total sums the total offered load on TOOLSNET from the offered load
** contributed by each individual host.
*/

#include "tap.h"

float Ro_total(rarr)
struct ratestats *rarr;          /* rarr contains the rates of each host */
{
     float acRo = 0.0;
     struct ratestats *r;

     for (r = rarr; r->sys ; r++)
     {
         if (r->Ro > 0)
           acRo += r->Ro;
     }
     return(acRo);
}
```

```
/* "%W% %G% */

#include "tap.h"
/* The purpose of addstats is to add the totals gleaned from the driver's
** counters to an accumulator, needed for further processing.
*/
zeroaccum(ac)
ac_allstats *ac;
{
    ac->rcvd_frames = 0;
    ac->xmit_frames = 0;
    ac->excess_collisions = 0;
    ac->rcvd_collision_fragments = 0;
    ac->lost_frames = 0;
    ac->crc_errors = 0;
    ac->align_errors = 0;
    ac->collisions = 0;
    ac->oow_collisions = 0;
    ac->acpt_multicast  = 0;
    ac->rej_multicast = 0;
    z_plen(ac);
    z_pno(ac);
} /* addstats */

z_plen(ac)
ac_allstats *ac;

    int i;

    for (i = 0; i < NSYS ; i++)
       ac->len_no[i].len = 0;



z_pno(ac)
ac_allstats *ac;

    int i;

    for (i = 0; i < NSYS; i++)
    ac->len_no[i].npkt = 0;
```

E.4 Shared code between doglobals and tnet week

```
/* Adjust_counters to reflect the sum total of the number of packets accumulat
*/

#include "tap.h"

adjust_counters(ac)
ac_allstats *ac;
{
    int i;
    ac->rcvd_frames = ac->xmit_frames = 0;

    for (i = 0; i < NSYS; i++)
        ac->rcvd_frames = ac->xmit_frames += ac->len_no[i].npkt;
}
```

```
/* "%W%" "%G%" */
/* The purpose of pdp_read is to read files from PDP 11/70'S that are storing
** TAP data for processing.  Seeing that doglobals runs on vaxes the data
** storage characteristics of the 11/70's have to be overcome someway.
** Remember: 11/70's place the low byte of the word first and the low 2 bytes
** of a long word first.
** Remember also that the VAX C compiler always pads out to a mod 4 word
** boundary when nesting structures and that the PDP pads out to a mod 2 bound
** These differences could have been overcome in a variety of different ways.
** I chose this way cause it was the easiest thing to do at the time that
** this problem came up (i.e. late).
** Had I had the forsight, I would have planned for this in the design.
** Ignorance and a bad memory are not blissful.
*/

#include <stdio.h>
#include "tap.h"
#define ps(string,dec) printf("%s : %d\n",string,dec)


#define TORCVD 38               /* To where transmitting packets begin. */
#define TOSD 94                 /* To where source and destination begins. */

static unsigned char buf[512];
static char file[]= "/D/usr/adm/blkdayread";
static unsigned char *bp, *tp;

pdp_getallstats(new,fp)
allstats *new;
FILE *fp;

    unsigned short mk_short();
    int i;
    unsigned short size = 0;
    int temp;
    char  b;
    char c;
    int a;

    bp = &buf[0];

    if(fread(&temp,2,1,fp) == 0)
    {
        new->stime[0] = NULL; /* This should be NULL but don't take chances. */
        return;
    }

    bp[0] = fgetc(fp);
    bp[1] = fgetc(fp);          /* Throw away extra char. */
    size = mk_short();
    /*
    printf("%d\n",size);
    */

    if(fread(buf,size - 4,1,fp) == 0)       /* buf[0] = sysname */
    {
```

```
        new->stime[0] = NULL; /* This should be NULL but don't take chances.
        return;
    }

    bp = &buf[0];


    new->sys = bp[0]; /*  Should be null 4 */
        /*
    printf("new->sys =  %c\n",new->sys);
        */

    for(i = 0 , bp = &buf[1]; i < NCHARS + 2; i++, bp++)
       new->stime[i] = *bp;      /* get stime */
    /* Look ahead
        for (i = 1, tp = bp; i < 15 ; i++, tp++)
            printf("%d\n",*tp);
        *** for debugging */


    new->drvrsts.intrlnsts.zero =   *bp;    /* hibyte */
    /*
    ps("zero",new->drvrsts.intrlnsts.zero);
    */

    bp++;

    new->drvrsts.intrlnsts.length = *bp; /* gets 62 */

    /*
    ps("length",new->drvrsts.intrlnsts.length);
    */

    bp++;                                   /* past 0 */

    /*
    for(i = 1, tp = bp; i < 6; tp++,i++)
        printf("%d",*tp);
    putchar('\n');
    */

    bp++;                                   /* past 0 */

    /* 2 7 1 ? ? ? unique ethernet address */

    for(i = 0; i < 6 ; i++, bp++)
        new->drvrsts.intrlnsts.phys_adrs[i] = *bp;
    /*
        printf("phys_addrs: %d",new->drvrsts.intrlnsts.phys_adrs[i] = *bp);
        putchar('\n');
        */

    bp  = &buf[TORCVD]; /* Don't mess around; go right to the stuff. */
    new->drvrsts.intrlnsts.rcvd_frames = mk_short();
    bp++;
    /*
```

```
    printf("rcvd = %d\n",new->drvrsts.intrlnsts.rcvd_frames);
    */


    new->drvrsts.intrlnsts.fifo_frames = mk_short();
        bp++;

    new->drvrsts.intrlnsts.xmit_frames = mk_short();
    bp++;

    new->drvrsts.intrlnsts.excess_collisions = mk_short();
    bp++;

    new->drvrsts.intrlnsts.rcvd_collision_fragments = mk_short();
    bp++;

    new->drvrsts.intrlnsts.lost_frames = mk_short();
    bp++;

    new->drvrsts.intrlnsts.acpt_multicast = mk_short();
    bp++;

    new->drvrsts.intrlnsts.rej_multicast = mk_short();
    bp++;

    new->drvrsts.intrlnsts.crc_errors = mk_short();
    bp++;


    new->drvrsts.intrlnsts.align_errors = mk_short();
    bp++;

    new->drvrsts.intrlnsts.collisions = mk_short();
    bp++;

    new->drvrsts.intrlnsts.oow_collisions = mk_short();

     /* Skip reserved, module_id, firmware id fields junk! */

    bp = &buf[TOSD];
    for (i = 0; i < NSYS; i++)  /* Deal with the long words on the ll */
    {
        unsigned short s1;
        unsigned short s2;

        new->drvrsts.len_no[i].npkt = mk_short();
        bp++;
        s1 = mk_short();
        bp++;
        s2 = mk_short();
        new->drvrsts.len_no[i].len = s1<<16 | s2;
        bp++;

    }/*endfor*/

  /*  If you are debugging then call:
```

```
   pdp_formwrite(new);
   */

}/* end*/
```

```
/* The purpose of this procedure is to convert the byte order of  pdp 11 bytes
** in  a character stream to their intended short integer representation.
** PDP's order there bytes "lo-hi".
*/
unsigned short mk_short()
{
    unsigned char pdplo;
    unsigned char pdphi;
    unsigned short c = 0;

    pdplo = *bp;
    bp++;
    pdphi = *bp;

    c |= (unsigned short)pdplo;

    c |= (unsigned short)pdphi << 8;

    return(c);
```

```
/* This procedure is left around for debugging. */

pdp_formwrite(stat)
allstats *stat;
{
    int i;

    char TIME[20];
    char *s;
    char *p = &TIME[0];

    for (s = &stat->stime[4]; s <= &stat->stime[4] + 14; s++)
    {
        *p++ = *s;
    }
    *p = NULL;

    printf("%s;%d;%d;%d;%d;%d;%d;%d;%d;", /* d;d;d;d; */

                /*stat->stime,
                */

                TIME,
                stat->drvrsts.intrlnsts.rcvd_frames,
                /*
                stat->drvrsts.intrlnsts.fifo_frames,
                */
                stat->drvrsts.intrlnsts.xmit_frames,
                stat->drvrsts.intrlnsts.excess_collisions,
                stat->drvrsts.intrlnsts.rcvd_collision_fragments,
                stat->drvrsts.intrlnsts.lost_frames,
                stat->drvrsts.intrlnsts.crc_errors,
                stat->drvrsts.intrlnsts.align_errors,
                stat->drvrsts.intrlnsts.collisions
                /*
                stat->drvrsts.intrlnsts.oow_collisions,
                stat->drvrsts.intrlnsts.acpt_multicast,
                stat->drvrsts.intrlnsts.rej_multicast
                */);


   for (i = 0; i < NSYS; i++)
       printf("%d;%ld;",stat->drvrsts.len_no[i].npkt,
                stat->drvrsts.len_no[i].len );
   putchar('\n');
   dbgms("leaving formwrite");
```

E.5 Tnet week code

```
/*****************************************************************************

The purpose of tnet_week() is to sum up the TOOLSNET activity for each system
and for TOOLSNET itself for a given work week.
The procedure pub_totals() in the file doglobals.c appends a day's
worth of each system's activity to the files:

            TAP_BLK_TOT_D   ( binary block i/o files)
        TAP_BLK_TOT_F
        TAP_BLK_TOT_H
        TAP_BLK_TOT_S
        TAP_BLK_TOT_V
        TAP_BLK_TOT_W
        TAP_BLK_TOT_X
        TAP_BLK_TOT_Z

        TAP_TOT_D       (ascii versions)
        TAP_TOT_F
        TAP_TOT_H
        TAP_TOT_S
        TAP_TOT_V
        TAP_TOT_W
        TAP_TOT_X
        TAP_TOT_Z.

Pub_totals() also appends a  days worth of TOTAL TOOLSNET activity to the files:
TNET_DAY_BLK_TOTAL and TNET_DAY_TOTAL.

Tnet_week() runs at the end of the week and totals up the contents of the *BLK*
files and appends the totals to the ascii non-BLK files so that people can read
them.

*****************************************************************************/
```

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/dir.h>
#include "tap.h"

#define STAR '*'
#define MAXLINE 256
#define MAXPATH 100
#define MAXFILES 200

char  targdir[] = "/usr/adm/";
char tap_blk_tot_sys[] = "TAP_BLK_TOT_@";
char tap_tot_sys[] = "TAP_TOT_@";
char tnet_day_blk_total[] =  "TNET_DAY_BLK_TOTAL";
char hosts[20];
char spooldirslash[MAXLINE];            /*  working directory   */
char cmd[MAXLINE];
int slength;
DIR *dirp;
struct direct *dp;
ac_allstats accum_week, *ac_week;

main()
{
    char filename[MAXPATH];
    char ascii_filename[MAXPATH];
    char *fn;
    char *ascii_fn, *get_ascii_fn();
    FILE *fp;
    int i;



    zeroaccum(&accum_week);
    ac_week = &accum_week;

    dirp = opendir(targdir);

    slength = strlen(tap_blk_tot_sys) - 1;
    /* Total TOOLSNET view as seen by each system. */

    for (dp =  readdir(dirp) ;   /* Find first system file and total */
        ((i = strncmp(dp->d_name,tap_blk_tot_sys,slength)) != 0)&&(dp !=NULL);
        dp = readdir(dirp)); /*endfor*/

    strcpy(filename,targdir);
    strcat(filename,dp->d_name);
    strcpy(ascii_filename,targdir);
    strcat(ascii_filename,get_ascii_fn(tap_tot_sys,dp->d_name));

    ac_week->sys = dp->d_name[strlen(dp->d_name) - 1];

    total_cols(ac_week,filename);
```

```
    pr_totals(ac_week,ascii_filename);


    do
    {
        dp =  readdir(dirp) ;   /* Find the rest of the files and total */
        if((i =strncmp(dp->d_name,tap_blk_tot_sys,slength)) == 0)
        {
                strcpy(filename,targdir);
                strcat(filename,dp->d_name);
                strcpy(ascii_filename,targdir);
                strcat(ascii_filename, get_ascii_fn(tap_tot_sys,dp->d_name));/*cho

                ac_week->sys = dp->d_name[strlen(dp->d_name) - 1];

                total_cols(ac_week,filename);

                pr_totals(ac_week,ascii_filename);
        }
    }while(dp != NULL);

    closedir(dirp);

    /* Total day totals into a week total.  for all of TOOLSNET. */

    dirp = opendir(targdir);
    while(strcmp(dp->d_name,tnet_day_blk_total) != 0)
    {
        dp = readdir(dirp);
    }
    strcpy(filename,targdir);
    strcat(filename,dp->d_name);
    strcpy(ascii_filename,targdir);
    strcat(ascii_filename,"TNET_DAY_TOTAL");
    total_cols(ac_week,filename);/* prints out total for week */
    pr_totals(ac_week,ascii_filename);
    closedir(dirp);

}/* endmain */


total_cols(ac,filename)
ac_allstats *ac;
char  *filename;
{
    ac_allstats new;   /* Will be reading files produced by pub_totals(). */
    int i;
    FILE *fp;
    fp = fopen(filename,"r");
    while((i = fread(&new,sizeof(new),1,fp)) != 0)
    {
       ac_ac_addstats(ac,&new);
    }

}
```

```
pr_totals(ac,filename)
ac_allstats *ac;
char  *filename;
{

    FILE *fp;

    fp = fopen(filename,"a");
    if(fp == NULL)
    {
        puts("pr_totals. Can't open");
        exit();
    }
    strcpy(ac->stime,"TOTAL");
    ac_formwrite(fp,ac);
    fclose(fp);
    zeroaccum(ac);
}

char * get_ascii_fn(ret_string,blockfile)
char *ret_string, *blockfile;
{
    char *r, *b;

    for(b = blockfile; *b != NULL ; b++);
    b--;
    for(r = ret_string; *r != NULL ; r++);
    r--;
    *r = *b;

    return(ret_string);
}
```

E.6 Stand-alone programs and shell scripts

```
#include <stdio.h>

#define BIGNUM 2147483647
/* This program averages the columns directed into it from stdin.  There is
** no header information.  This allows the user to use this with shell
** scripts.  It is mainly intended as a viewing convenience.
** Currently it is called by the shell scripts totavgr and totavgd.
**
**              number of packets;number of bytes;
**              integer;integer;
**              integer;integer;
**              integer;integer;
**              etc;etc;
**
*/

main()
{
    char line[512];                     /* for getting rid of title. */
    long npkt, nbyte, npkt_sum = 0, nbyte_sum = 0;
    int t;


    gets(line);         /* get rid of title */



    while ((t = scanf("%ld;%ld;",&npkt,&nbyte)) != EOF)
    {

        npkt_sum += npkt;
        nbyte_sum += nbyte;

        if (nbyte_sum >= BIGNUM - 1000)
        {
            printf("Warning: Close to overflow. Total of column thus far: ");
            printf("%d\n",nbyte_sum);
            exit();
        }
    }
    if(npkt_sum)
       printf("npkt_sum: %ld\tnbyte_sum: %ld\taverage: %f",
            npkt_sum,nbyte_sum,(double)(nbyte_sum/npkt_sum));
}
```

```
#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/toolsnet.h>
#define O_RDWR 002

main(argc,argv)
char **argv;
int argc;
{
        FILE *fp;
        register status= -1;
        register fd;



        fd = open("/dev/icg", O_RDWR);        /* Call driver for read */
        if ( fd < 0 ){
             printf("can't open\n");
             exit(1);
            }

        if((argv[1][0] == '-') && (argv[1][1] == 'o'))
        {
            /* status = ioctl(fd, (ON_COUNTERS|IOC_VOID), (char *) 0); */
            status = ioctl(fd, (ON_COUNTERS), (char *) 0);
            printf("Turned source and destination counters on.\n");
            printf("If you want them off, just type <sdwatch>\n");
        }
        else
        {
            /* status = ioctl(fd, (OFF_COUNTERS|IOC_VOID), (char *) 0); */
            status = ioctl(fd, (OFF_COUNTERS), (char *) 0);
            printf("Turned source and destination counters off.\n");
            printf("If you want them on, just type <sdwatch -on>\n");
        }

}
```

```
#include <stdio.h>

#define BIGNUM 2147483647
/* This program totals the column directed into it from stdin.  There is
** no header information.  This allows the user to use this with shell
** scripts.  It is currently used with the totavg family of programs.
** Since it only reads on column, it can be used with any column
** in DBMS format.  It expect the column to be in this format:
**
**              title;
**              integer;
**              integer;
**              integer;
**              etc;
**
*/

main()
{
    char line[512];                     /* for getting rid of title. */
    long col,col_sum = 0;
    int t;


    gets(line);         /* get rid of title */



    while ((t = scanf("%ld;",&col)) != EOF)
    {
        col_sum += col;
        if (col_sum >= BIGNUM - 1000)
        {
            printf("Warning: Close to overflow. Total of column thus far: ");
            printf("%d\n",col_sum);
        }
    }
    printf("%ld",col_sum);
}
```

```
#include <stdio.h>

/* This program truncates a "dayread" file that is sorted by number of
** transmissions.  This will help discern the NMEASUREMENTS that had the
** most traffic patterns for a given day.
** One extra line must be alloted for the the title: xf, for transmitted
** frames.
**
** This program is currently called by /usr/tnacct/cleanup.
**
** The  command line should look something like this:
** trunc /usr/adm/dayread 10
**
*/
main(argc,argv)
int argc;
char **argv;
{

    char line[512];
    int i;
    int NMEASUREMENTS;
    FILE *fp;

    if(argc != 3)
    {
        puts("Bad format");
        puts("Type in the following form:");
        puts("\ntrunc filename nlines");
        exit();
    }

    fp = fopen(argv[1], "r");
    if (fp == NULL)
    {
        puts("can't open");
        puts(argv[1]);
        exit();
    }


    NMEASUREMENTS = atoi(argv[2]);
    while(fgets(line,512,fp) &&  NMEASUREMENTS--)
    {
        printf("%s",line);
    }
}
```

```
/* static char filename[] = " %W% %G% "; */
#include "tap.h"

/* The purpose of this program is to extract the counters collected
** in the driver, send them to the screen and save them in tmpfl when
** not just clearing in the morning.
** The idea is to get the user to type: "vdr" and have that stats saved and
** to have cron type vdr -<whatever_it_feels_like> to clear the controller
** in the morning and not save the counters in tmpfl.
*/

allstats s;
char tmpfl[] = "/usr/adm/tmpfl";
main(argc,argv)
int argc;
char **argv;
{
        FILE *fp;
        register i=0;
        register fd;

        long time();
        long thetime, time();
        char *ctime();


        fd = open("/dev/icg", 0);       /* Call driver for read */
        if ( fd < 0 ){
             printf("can't open\n");
             exit(1);
            }

        read(fd, &s.drvrsts, sizeof(s.drvrsts));

        close(fd);
        strcpy(s.stime,ctime(&thetime));

        if(argc == 1 )          /* Any argument will abort write to tmpfl. */
        {

            fp = fopen(tmpfl,"a");
                i = 0;

            while (fp < 0){

                if (++i == 20){
                      printf("can't open\n");
                      exit(1);
                }/*endif*/
            }/*endwhile*/

            fwrite(&s,sizeof(s),1,fp);
            fclose(fp);

        }/*endif*/
```

```
        put();
}

put()
{
    long time(),thetime;
    char *ctime();
    float avg();
    thetime = time(0);

    printf("Time decimal = %d\n", thetime);
    printf("%s", ctime(&thetime));



    printf("\tzero = %d(0), length = %d(62), phys adrs = %x %x %x %x %x %x\n",
                s.drvrsts.intrlnsts.zero, s.drvrsts.intrlnsts.length,
                s.drvrsts.intrlnsts.phys_adrs[0], s.drvrsts.intrlnsts.phys_adrs[1]
                s.drvrsts.intrlnsts.phys_adrs[2], s.drvrsts.intrlnsts.phys_adrs[3]
                s.drvrsts.intrlnsts.phys_adrs[4], s.drvrsts.intrlnsts.phys_adrs[5]
    printf("\tframes received = %u\t\t\tframes in receive fifo = %d\n",
                s.drvrsts.intrlnsts.rcvd_frames, s.drvrsts.intrlnsts.fifo_frames);
    printf("\tframes transmitted =%u\t\t\texcess collisions = %d\n",
                s.drvrsts.intrlnsts.xmit_frames, s.drvrsts.intrlnsts.excess_collis
    printf("\tcollision fragments received = %d\tframes lost %d times\n",
                s.drvrsts.intrlnsts.rcvd_collision_fragments, s.drvrsts.intrlnsts.
    printf("\tmulticast frames: accepted = %d\t\tframes rejected = %d\n",
                s.drvrsts.intrlnsts.acpt_multicast, s.drvrsts.intrlnsts.rej_multic
    printf("\terror frames received: crc = %d\t\talignment = %d\n",
                s.drvrsts.intrlnsts.crc_errors, s.drvrsts.intrlnsts.align_errors);
    printf("\tcollisions = %d\t\t\tout of window collisions = %d\n",
                s.drvrsts.intrlnsts.collisions, s.drvrsts.intrlnsts.oow_collisions
    printf("\treserved = %x %x %x %x %x %x %x %x\n",
                s.drvrsts.intrlnsts.reserved[0], s.drvrsts.intrlnsts.reserved[1],
                s.drvrsts.intrlnsts.reserved[2], s.drvrsts.intrlnsts.reserved[3],
                s.drvrsts.intrlnsts.reserved[4], s.drvrsts.intrlnsts.reserved[5],
                s.drvrsts.intrlnsts.reserved[6], s.drvrsts.intrlnsts.reserved[7]);
    printf("\tmodule_id = %c %c %c %c %c %c %c %c\n",
                s.drvrsts.intrlnsts.module_id[0], s.drvrsts.intrlnsts.module_id[1]
                s.drvrsts.intrlnsts.module_id[2], s.drvrsts.intrlnsts.module_id[3]
                s.drvrsts.intrlnsts.module_id[4], s.drvrsts.intrlnsts.module_id[5
                s.drvrsts.intrlnsts.module_id[6], s.drvrsts.intrlnsts.module_id[7
    printf("\tfirmware_id = %c %c %c %c %c %c %c %c\n\n",
                s.drvrsts.intrlnsts.firmware_id[0], s.drvrsts.intrlnsts.firmware_
                s.drvrsts.intrlnsts.firmware_id[2],
                s.drvrsts.intrlnsts.firmware_id[3], s.drvrsts.intrlnsts.firmware_
                s.drvrsts.intrlnsts.firmware_id[5],
                s.drvrsts.intrlnsts.firmware_id[6], s.drvrsts.intrlnsts.firmware_

ifdef VAX /* The collectors on the Vaxes do not see ACKS, so double. */
    printf("\tNo. pkts from system D: %d\tTotal bytes: %ld\tAVG: %.0f\n",
            s.drvrsts.len_no[0].npkt * 2, s.drvrsts.len_no[0].len * 2, avg(0) );
    printf("\tNo. pkts from system F: %d\tTotal bytes: %ld\tAVG: %.0f\n",
            s.drvrsts.len_no[1].npkt * 2, s.drvrsts.len_no[1].len * 2, avg(1) );
    printf("\tNo. pkts from system H: %d\tTotal bytes: %ld\tAVG: %.0f\n",
```

```
            s.drvrsts.len_no[2].npkt * 2, s.drvrsts.len_no[2].len * 2, avg(2) );
    printf("\tNo. pkts from system V: %d\tTotal bytes: %ld\tAVG: %.0f\n",
            s.drvrsts.len_no[3].npkt * 2, s.drvrsts.len_no[3].len * 2, avg(3) );
    printf("\tNo. pkts from system W: %d\tTotal bytes: %ld\tAVG: %.0f\n",
            s.drvrsts.len_no[4].npkt * 2, s.drvrsts.len_no[4].len * 2, avg(4) );
    printf("\tNo. pkts from system X: %d\tTotal bytes: %ld\tAVG: %.0f\n",
            s.drvrsts.len_no[5].npkt * 2, s.drvrsts.len_no[5].len  * 2, avg(5));
    printf("\tNo. pkts from system Z: %d\tTotal bytes: %ld\tAVG: %.0f\n",
            s.drvrsts.len_no[6].npkt * 2, s.drvrsts.len_no[6].len  * 2, avg(6));
    printf("\tNo. pkts from system S: %d\tTotal bytes: %ld\tAVG: %.0f\n",
            s.drvrsts.len_no[7].npkt * 2, s.drvrsts.len_no[7].len * 2, avg(7) );
#endif

#ifdef PDP /* The collectors on the PDPs see both DATA and ACKS. DON'T DOUBLE */

    printf("\tNo. pkts from system D: %d\tTotal bytes: %ld\tAVG: %.0f\n",
            s.drvrsts.len_no[0].npkt, s.drvrsts.len_no[0].len, avg(0) );
    printf("\tNo. pkts from system F: %d\tTotal bytes: %ld\tAVG: %.0f\n",
            s.drvrsts.len_no[1].npkt , s.drvrsts.len_no[1].len , avg(1) );
    printf("\tNo. pkts from system H: %d\tTotal bytes: %ld\tAVG: %.0f\n",
            s.drvrsts.len_no[2].npkt , s.drvrsts.len_no[2].len , avg(2) );
    printf("\tNo. pkts from system V: %d\tTotal bytes: %ld\tAVG: %.0f\n",
            s.drvrsts.len_no[3].npkt , s.drvrsts.len_no[3].len , avg(3) );
    printf("\tNo. pkts from system W: %d\tTotal bytes: %ld\tAVG: %.0f\n",
            s.drvrsts.len_no[4].npkt , s.drvrsts.len_no[4].len , avg(4) );
    printf("\tNo. pkts from system X: %d\tTotal bytes: %ld\tAVG: %.0f\n",
            s.drvrsts.len_no[5].npkt , s.drvrsts.len_no[5].len  , avg(5));
    printf("\tNo. pkts from system Z: %d\tTotal bytes: %ld\tAVG: %.0f\n",
            s.drvrsts.len_no[6].npkt , s.drvrsts.len_no[6].len  , avg(6));
    printf("\tNo. pkts from system S: %d\tTotal bytes: %ld\tAVG: %.0f\n",
            s.drvrsts.len_no[7].npkt, s.drvrsts.len_no[7].len , avg(7) );
#endif


float avg(i)
int i;

    if(s.drvrsts.len_no[i].npkt > 0)
        return(s.drvrsts.len_no[i].len  / s.drvrsts.len_no[i].npkt);
    return(0.0);
```

```
# Develop history files to establish traffic patterns, and keep sometype of
# running network history.
#
#
#
echo "cat /usr/adm/dayread > /usr/tnacct/dayread.yesterday"
cat /usr/adm/dayread > /usr/tnacct/dayread.yesterday
echo "cat /usr/adm/blkdayread > /usr/tnacct/blkdayread.yesterday"
cat /usr/adm/blkdayread > /usr/tnacct/blkdayread.yesterday
#
# If you decide to archive entire dayread files for analysis of longer
# periods of time then do something like the next few commented lines.
#
# cat /usr/adm/dayread >> /usr/tnacct/dayread.accum
# cat /usr/adm/blkdayread >> /usr/tnacct/blkdayread.accum
#
# sort and append dayread files by number of tranmsissions
#
a=/usr/adm
b=/usr/tnacct
for i in S V W X Z D F H
do
    cat hdl /$i$a/dayread | column time xf \
                | dbsort -r -n xf | pprint > $b/traffic
    echo " " >> $b/peaktimes
    echo "Peaktime traffic for system: $i" >> $b/peaktimes
    $b/trunc $b/traffic 30 >> $b/peaktimes
done
echo " " >> $b/peaktimes
echo " " >> $b/peaktimes
mail sorrent < peaktimes
spool -qb peaktimes
#
# Do this only on system S
echo "/tools/sys/sorrentino/th.code/dgl d >> /usr/adm/TNET_GLOBS"
date >> /usr/adm/TNET_GLOBS
/tools/sys/sorrentino/th.code/dgl d >> /usr/adm/TNET_GLOBS
echo " " >> /usr/adm/TNET_GLOBS
mail sorrent < /usr/adm/TNET_GLOBS
#
rm /usr/adm/*dayread
rm /usr/adm/*recentread
rm /usr/adm/tmpfl
```

```
echo
echo "<<<<<< TOOLSNET TRAFFIC FROM SYSTEM $1 TO OTHER SYSTEMS >>>>>>"
echo
cp  /usr/tnacct/hkey  /usr/tnacct/hkey1
cat  /usr/tnacct/hd1 /$1/usr/adm/dayread | pprint >> /usr/tnacct/hkey1
cat  /usr/tnacct/hkey1 | cat
```

```
echo
echo "<<<<<< TOOLSNET TRAFFIC FROM SYSTEM $1 TO OTHER SYSTEMS >>>>>>"
echo
echo "Print out interlan controller data from dayread file specified."
echo "cat /usr/tnacct/hdl /$1/usr/adm/dayread | pprint >>  /usr/tnacct/hkey1"
cp  /usr/tnacct/hkey  /usr/tnacct/hkey1
cat  /usr/tnacct/hdl /$1/usr/adm/dayread | column -x Dn Dl Hn Hl Fn Fl Vn Vl\
  Wn Wl Xn Xl Zn Zl Sn Sl | pprint >>  /usr/tnacct/hkey1
cat  /usr/tnacct/hkey1
```

```
echo
echo "<<<<<< TOOLSNET TRAFFIC FROM SYSTEM $1 TO OTHER SYSTEMS >>>>>>"
echo
echo "Print out source and destination data from dayread file specified."
echo
echo "cat /usr/tnacct/hd1 /$1/usr/adm/dayread | pprint >>  /usr/tnacct/hkey1"
cp  /usr/tnacct/hkeysd  /usr/tnacct/hkey1
cat  /usr/tnacct/hd1 /$1/usr/adm/dayread | column -x ec rc 1f ce ae co \
 | /usr/cg/bin/pprint >>  /usr/tnacct/hkey1
cat  /usr/tnacct/hkey1
```

```
echo
echo "<<<<<< TOOLSNET TRAFFIC FROM SYSTEM $1 TO OTHER SYSTEMS >>>>>>"
echo
cp  /usr/tnacct/hkey  /usr/tnacct/hkey1
cat  /usr/tnacct/hd1 /$1/usr/adm/recentread | pprint >> /usr/tnacct/hkey1
cat  /usr/tnacct/hkey1 | cat
```

```
echo
echo "<<<<<< TOOLSNET TRAFFIC FROM SYSTEM $1 TO OTHER SYSTEMS >>>>>>"
echo
echo "Print out interlan controller data from recentread file specified."
echo "cat /usr/tnacct/hdl /$1/usr/adm/recentread | pprint >>  /usr/tnacct/hkey1"
cp  /usr/tnacct/hkey  /usr/tnacct/hkey1
cat  /usr/tnacct/hdl /$1/usr/adm/recentread | column -x Dn Dl Hn Hl Fn Fl Vn Vl\
  Wn Wl Xn Xl Zn Zl Sn Sl | pprint >>  /usr/tnacct/hkey1
cat  /usr/tnacct/hkey1
```

```
echo
echo "<<<<<< TOOLSNET TRAFFIC FROM SYSTEM $1 TO OTHER SYSTEMS >>>>>>"
echo
echo "Print out source and destination data from recentread file specified."
echo "cat /usr/tnacct/hdl /$1/usr/adm/recentread | pprint >>  /usr/tnacct/hkey1"
cp  /usr/tnacct/hkeysd  /usr/tnacct/hkey1
cat  /usr/tnacct/hdl /$1/usr/adm/recentread | column -x ec rc lf ce ae co \
 | /usr/cg/bin/pprint >>  /usr/tnacct/hkey1
cat  /usr/tnacct/hkey1
```

```
echo
echo "<<<<<< TOOLSNET TRAFFIC FROM SYSTEM $1 TO OTHER SYSTEMS >>>>>>"
echo
echo ":::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::"
echo " Values for Systems (Dn, Dl, Hn, etc., ) are in the 1000's."
echo ":::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::"
echo
cp  /usr/tnacct/hkey  /usr/tnacct/hkey1
cat  /usr/tnacct/hdl /usr/adm/TAP_TOT_$1 | pprint >> /usr/tnacct/hkey1
cat  /usr/tnacct/hkey1 | cat
```

```
cp  /usr/tnacct/hkey  /usr/tnacct/hkey1
echo "Print out interlan controller data from TAP_TOT_$1 file specified."
cat  /usr/tnacct/hd1 /usr/adm/TAP_TOT_$1 | column -x Dn D1 Hn H1 Fn F1 Vn V1\
  Wn W1 Xn X1 Zn Z1 Sn S1 | pprint >>  /usr/tnacct/hkey1
cat  /usr/tnacct/hkey1
```

```
#
echo "Print out source and destination data from TAP_TOT_$1 file specified."
echo
echo "<<<<<<<<<<<<<<<< Traffic to Systems is in 1000's >>>>>>>>>>>>>>>>>>>>>>"
echo
cp  /usr/tnacct/hkeysd  /usr/tnacct/hkey1
cat  /usr/tnacct/hdl /usr/adm/TAP_TOT_$1 | column -x xf rf ec rc lf ce ae co \
 | /usr/cg/bin/pprint >>  /usr/tnacct/hkey1
cat  /usr/tnacct/hkey1
```

```
echo
echo "<<<<<< PRINT OUT CONTENTS OF /USR/ADM/TNET_DAY_TOTAL >>>>> "
echo "<<<<<< THIS IS A PICTURE OF TOTAL TOOLSNET ACTIVITY FOR EACH DAY >>>>>>
echo
echo ":::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::"
echo " Values for Systems (Dn, Dl, Hn, etc., ) are in the 1000's."
echo ":::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::"
echo
cp  /usr/tnacct/hkey  /usr/tnacct/hkeyl
cat  /usr/tnacct/hdl /usr/adm/TNET_DAY_TOTAL | pprint >> /usr/tnacct/hkeyl
cat  /usr/tnacct/hkeyl | cat
```

```
cp  /usr/tnacct/hkey  /usr/tnacct/hkey1
echo "Print out interlan controller data from TNET_DAY_TOTAL file specified."
echo "cat /usr/tnacct/hdl TNET_DAY_TOTAL | pprint >>  /usr/tnacct/hkey1"
cat  /usr/tnacct/hdl /usr/adm/TNET_DAY_TOTAL | column -x Dn Dl Hn Hl Fn Fl Vn
  Wn Wl Xn Xl Zn Zl Sn Sl | pprint >>  /usr/tnacct/hkey1
cat  /usr/tnacct/hkey1
```

```
cp  /usr/tnacct/hkeysd  /usr/tnacct/hkey1
echo "Print out source and destination data from TNET_DAY_TOTAL file specified
echo
echo "<<<<<<<<<<<<<<<< Traffic to Systems is in 1000's >>>>>>>>>>>>>>>>>>>>>>"
echo
cat  /usr/tnacct/hdl /usr/adm/TNET_DAY_TOTAL | column -x xf rf ec rc lf ce ae
 | /usr/cg/bin/pprint >>  /usr/tnacct/hkey1
cat  /usr/tnacct/hkey1
```

```
l=l
n=n
u=/usr/tnacct
for i in $*
do
   echo

   echo

   echo

   echo

   echo "Total transmits, packets and bytes transmitted by System $i"
   echo "=========================================================="
   echo
   cp /$i/usr/adm/dayread  /usr/tnacct/tl
   cat /$u/hdl /$u/tl | column time | /usr/tnacct/t_range
   echo
   echo "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
   echo "Total number of packets transmitted by system $i:"
   cat /$u/hdl /$u/tl | column xf | /usr/tnacct/tdc
   echo
   for j in D F H V W X Z
   do
        echo "*****************************************************"
        echo "Total number of packets, bytes, and averages\
                  transmitted to $j by $i:"
        cat /$u/hdl /$u/tl | column $j$n $j$l | /usr/tnacct/average
        echo
   done
   echo "
```

```
l=l
n=n
u=/usr/tnacct
for i in $*
do
   echo

   echo

   echo

   echo

   echo "Total transmits, packets and bytes transmitted by System $i"
   echo "============================================================"
   echo
   cp /$i/usr/adm/recentread  /$u/tl
   cat /$u/hdl /$u/tl | column time | /usr/tnacct/t_range
   echo
   echo "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
   echo "Total number of packets transmitted by system $i:"
   cat /$u/hdl /$u/tl | column xf | /usr/tnacct/tdc
   echo
   for j in D F H V W X Z
   do
        echo "*****************************************************"
        echo "Total number of packets, bytes, and averages\
                transmitted to $j by $i:"
        cat /$u/hdl /$u/tl | column $j$n $j$l | /usr/tnacct/average
        echo
   done
   echo "
```