The IEEE 802.4 Token Bus --
An Introduction and Performance Analysis

Catherine F. Summers, M.S.
Alfred C. Weaver, Ph.D.

Computer Science Report No. TR-85-19
August 26, 1985

The IEEE 802.4 Token Bus — An Introduction and Performance Analysis

Catherine F. Summers, M.S.
Alfred C. Weaver, Ph.D.
Department of Computer Science
University of Virginia

Abstract

We present a brief tutorial on the new IEEE 802.4 standard for token bus networks, followed by a performance analysis based upon simulation studies. We show how total bus capacity is divided among data throughput, token traffic, and propagation delays. We show the relative contributions of access delay and queueing delay to total message delivery time as functions of the network's offered load. The effects of message size and the number of active stations on message delivery times are also investigated. As the token cycle time increases beyond the *Target_Rotation_Times* for the non-*Synchronous access_classes*, service to the lower priority classes is curtailed; we present a formula which can be used to identify the offered load at which the transition (from normal to curtailed service) begins.

This paper has been submitted for publication in the *IEEE Transactions on Communications*.

1. The IEEE 802.4 Token Bus

The IEEE has established a standard for local area networks (LANs) that use an explicit token passing scheme to control access on a bus topology network. Known as IEEE 802.4 [IEEE 802.4, 85], the standard specifies the actions and services of the *Medium_Access_Controller*, or MAC, which is the software interface between the data link and physical layers of the ISO OSI model [Zimmerman, 80]. Words in this paper that are italicized are 802.4 reserved words.

While the physical topology is a bus, 802.4 creates a logical ring of active stations through the token passing process. 802.4 offers four priority classes for message transmissions and permits bounded message delivery times. The individual stations in an 802.4 network are free to leave and join the token passing ring as dictated by their traffic or station management decisions. The standard describes a robust protocol able to withstand the loss or duplication of tokens and the failure or disconnection of stations.

1.1. General Operation Characteristics

The token in an 802.4 network is an explicit message of at least 96 bits for networks using sixteen bit addresses and at least 160 bits for networks using forty-eight bit addresses. The MAC variable *NS* (for next station) is used by each station to maintain the address of its successor in the logical ring, and that address is loaded into the token's destination address field. The token consists of one octet (8 bits) for the start delimiter followed by one octet of control information identifying the message as a token; two or six octets for each of the destination and source addresses; four octets for the frame check sequence; and one octet for the end delimiter. A station can optionally add octets of data between the source address and the frame check sequence. The maximum size of any message in an 802.4 network is 8191 octets, excluding the start and end delimiters.

The station holding the token becomes the temporary master of the network. A series of timers, described below, limits the amount of time that a station can hold the token. During that interval, the token holder can transmit messages or poll other stations for

messages or acknowledgements. If the station has no use for the token, it will forward the token to its successor. After the transmission of a data message, the standard specifies that the MAC will wait an interframe gap of 2 microseconds before it begins the next transmission or passes the token; the interframe gap is needed to allow receiving stations time to process the previous message.

The token is passed from station to station in descending address order. The lowest addressed station passes the token to the active station with the largest address to close the logical ring. A station does not have to be an active member of the token passing ring to receive transmissions or to respond to queries directed to it by the current token holder.

1.2. Priority and the Access_Classes

802.4 offers four *access_classes* for prioritizing message transmissions. In decreasing priority order the *access_classes* are class six, the *Synchronous access_class*, class four, the *Urgent_Asynchronous access_class*, class two, the *Normal_Asynchronous access_class*, and class zero, the *Time_Available access_class*. An 802.4 station must offer either all of the *access_classes*, or else all transmissions must be routed through the *Synchronous access_class*. A station that does use the priority option must implement a queue for messages of each of the individual *access_classes*, and it must provide three *token_rotation_timers* for the *Urgent_Asynchronous*, *Normal_Asynchronous*, and *Time_Available access_classes*. These are count-down timers loaded with a station-dependent value, namely the *Target_Rotation_Time* (*TRT*) for each *access_class*, every time the station receives the token.

Whenever a station receives the token, it is guaranteed a certain amount of time for serving messages at the *Synchronous access_class*. This interval is a system-wide parameter known as the *High_Priority_Token_Hold_Time* (*HPTHT*). When a station receives the token it enters the *Use_Token* state and loads its *token_hold_timer* with the *HPTHT*. If there are any messages enqueued for the *Synchronous access_class*, the station begins to transmit those messages. If, after the completion of a transmission, the *token_hold_timer* has expired or the queue has been emptied, the station enters the *Check_Access_Class* state, and performs the

following procedure: if the current *access_class* is the *Time_Available access_class* or the station is not implementing the priority option, the station will enter the *Pass_Token* state; otherwise it will (1) decrement the current *access_class*; (2) reload the *token_hold_timer* with the *token_rotation_timer* for the new *access_class*; (3) reload the *token_rotation_timer* with the *Target_Rotation_Time* for this class; (4) return to the *Use_Token* state. The *TRT* values used to enforce priority operation and to allow service at the asynchronous *access_classes* are discussed in Section 6.

1.3. Logical Ring Membership

An 802.4 network usually consists of two or more stations connected in a logical ring. Generally, if a station believes itself to be the only member of a token passing ring, 802.4 assumes that the station is in error. Stations can leave and join the ring dynamically, and any station must be able to patch the ring around a failed neighbor. This section discusses dynamic station membership leaving the error conditions mentioned above for the next section.

Stations which are active members of the token passing ring maintain an *inter_solicit_count* which controls the opening of *response_windows*. Before passing the token, the station checks the *inter_solicit_count* and, if it is zero, opens a *response_window* which allows any station whose address is between the token holder's and the token holder's successor's addresses to petition for admittance to the token passing ring. If no stations respond during the open *response_window* the token will be passed to the token holder's successor and the token holder will reset the *inter_solicit_count*. If a single station responds during the *response_window* then the token holder will make the responding station its new successor and pass the token to the new ring member. If more than one station attempts to respond to the solicitation to join the ring, 802.4 uses an algorithm that will sequence through the address bits of the contending stations until one station has successfully responded or all the address bits have been examined (see section 6.5 of the standard). If multiple stations remain in contention then they have duplicate addresses; this error condi-

tion is then reported to the station management software, and the stations enter the *OffLine* state. When the token holder does receive a successful *Set_Successor* frame the station that was the source of the message will be patched into the ring, and the token holder will not reset the *inter_solicit_count* so that on the next token cycle it will again open a *response_window*.

Each station has two Boolean variables, *in_ring_desired* and *any_send_pending*, which are used to determine if the station should leave the token passing ring. The *in_ring_desired* variable is set by the station management software and as long as it remains true a station will remain an active participant of the token passing ring. The second variable, *any_send_pending*, is true whenever the MAC has any message in any of the *access_class* queues. A station that is a member of the token passing ring with *in_ring_desired* false but *any_send_pending* true will remain in the ring until it has emptied its queues.

When a station decides to leave the token passing ring there are two possible methods to effect the departure. The most drastic method is for the station to ignore the next token that is passed to it on the next token rotation. This method uses the error recovery mechanism to patch the departing station out of the ring. The more graceful method requires the station to wait until it receives its next token; as the token holder, the station transmits a *Set_Successor* frame to the preceding station with the token holder's *NS* (next station address) as the data field of the message.

1.4. Error Recovery

The 802.4 standard lists the errors that a MAC must be able to handle: lost or multiple tokens, a token-pass failure, a deaf station, and stations with duplicate addresses (described in the previous section). One obvious type of error, the corruption of a data message, is not handled by the MAC layer; the reception of a bad message is reported to the LLC layer, (LLC is the *Logical_Link_Control* standard, IEEE 802.2) and it is the responsibility of the LLC layer to deal with the error by requesting retransmission or whatever other algorithm it wishes to employ.

1.4.1. Lost or Multiple Tokens

For the correct operation of the token passing protocol there has to be one and only one token in the network. If there is no token then no station would be able to transmit; if there is more than one token multiple stations could attempt to transmit at the same time, losing messages through collisions.

Each station starts its *bus_idle_timer* whenever it is in the *Idle* station state and it senses an idle bus. If the *bus_idle_timer* expires, the station is in the *Idle* state, the bus is idle, and the station has messages to send or it wants to be a ring member and is not the sole active station, the station will transmit a *Claim_Token* frame and enter the *Claim_Token* state. A station in the *Claim_Token* state will, if the bus is quiet and the *claim_pass_count* is less than the *max_pass_count*, increment the *claim_pass_count*, transmit a *Claim_Token* frame with a data unit 0, 2, 4, or 6 slottimes long, (see section 6.1.9 of the standard for definition of slottime) and wait 1 slottime before repeating the process. If the station senses another station transmitting when the *claim_timer* expires it has lost the contention for the token and returns to the *Idle* state. If the *claim_pass_count* equals the *max_pass_count* the station has won the contention process and enters the *Use_Token* state.

The station with the lowest address sets the *bus_idle_timer* to six slottimes; all other stations wait seven slottimes. Since waiting stations drop out of the contention process when they detect other transmissions, the lowest addressed station should win the contention for the token on network initialization.

1.4.2. Token-Pass Failure

When a station transmits a token to its successor, it starts the *token_pass_timer* and enters the *Check_Token_Pass* state. If the station hears a transmission from another station before the timer expires, the station assumes that its successor has received the token, and the station enters the *Idle* state. If the timer expires and the station is still in the *Check_Token_Pass* state, it will assume that the token pass has failed. If the failed attempt was the first attempt at passing the token the station will transmit another token to its

successor, restart the *token_pass_timer*, and reenter the *Check_Token_Pass* state. When the second attempt at passing the token has failed the station will enter the *Pass_Token* state and try to reconnect the token passing ring.

After two attempts at passing the token to the successor station, the token holder first must evaluate the state of the bus to determine the course of action needed to repair the ring. A station seeking to repair the ring after two failed token passes has not heard any valid messages from any other stations since it first started the token passing process; if it had it would not be in this state. If the station senses a non-idle bus, then it assumes that its receiver is at fault (valid messages are being transmitted but not received by this station), and enters the *Idle* state.

A station that has had two token passes fail and that senses an idle bus will first assume that its successor has failed or has left the token passing ring without notifying this station. The station will transmit a *Who_Follows* frame with its failed successor's address, open three *response_windows*, and enter the *Await_Response* state. If the successor to the failed station responds with a *Set_Successor* frame the token holder sets its *NS* to the source address of the received *Set_Successor* frame and passes the token to the failed station's successor, thereby patching the failed station out of the ring. If the token holder does not get a response to the *Who_Follows* frame it will retransmit the frame.

When two attempts to elicit a response from the successor to the failed station have failed, the token holder will transmit a *Solicit_Any* frame to allow any other active station to respond with a *Set_Successor* frame and reconnect the ring. If a single valid response to the *Solicit_Any* frame is received the token holder will pass the token to the responder. If more than one station responds to the frame the resolution process described previously in section 1.4.1 is used to resolve the contention. Stations which were ring members between the token holder and its new successor will have to rejoin the ring as response windows are opened. If no response to the *Solicit_Any* frame is received the token holder assumes that it has a faulty receiver; the faulty receiver is reported to the LLC layer, and the sta-

tion enters the *OffLine* state.

2. Simulation Studies

A Pascal program that simulates the actions and state transitions of user configurable 802.4 networks was used to study the performance of 802.4 and to study the effects of varying important configuration parameters. The simulator was designed using Draft E of the 802.4 standard.

Unless otherwise noted, all simulations used the same basic configuration: an error-free 10Mbps bus, 96 bit tokens, functionally infinite HPTHT (10 secs), all traffic in the *Synchronous access_class* only, a logical ring of 64 stations with identical message creation rates and constant message sizes of 160 data bits (for a total of 256 bits in the frame), and all stations are always members of the token passing ring. Offered load was varied from 10 to 95 percent (increasing in steps of 5 percent per simulation) of the bus capacity by increasing the message creation rate λ. For example, to achieve an offered load of 10 percent of the bus capacity with the above configuration

$$\lambda = 0.10 \cdot \frac{10{,}000{,}000}{64 \cdot 256} \approx 61 \text{ messages per second.}$$

3. Base Configuration Results

A base configuration of 64 stations with identical Poisson message arrival rates and identical 256 bit message frames (160 data bits), all at the *Synchronous access_class*, is used to examine relationships between the offered load and three performance metrics: message throughput, average delivery time, and average token cycle time.

3.1. Utilization

There are four components of bus utilization: message throughput, token transmissions, propagation delay, and overhead such as protocol messages, corrupted messages, and idle periods during response windows. Since any portion of the bus capacity not utilized by message transmissions is, by definition, filled by the other three components, the sum of the

four components always equals bus capacity. The overhead component is generally negligible. Message throughput is defined to be the number of data bits transmitted per bit time where bit time is the inverse of the LAN data rate. The throughput can be measured by dividing the total number of bits transmitted, including all address and framing bits, by the product of the LAN data rate and the elapsed time. Message throughput is always bounded by the minimum of the offered load and the bus capacity. Two factors decrease message throughput: token transmissions and message propagation delays. Both consume network bandwidth which could otherwise be used for data (given sufficient offered load).

Figure 1 illustrates how the bus capacity is allocated for a given offered load. For the base configuration an aggregate offered load of fifty percent uses fifty percent of the bus capacity to carry messages (i.e., all data gets transmitted), thirty percent to carry tokens, and twenty percent is consumed by message and token propagation time. The dashed line (green) indicates the maximum message throughput for a given offered load. The solid line with asterisks (black) shows that actual message throughput is equal to the maximum possible throughput until offered load approaches bus capacity. The dotted line with circles (red) shows that as data message traffic increases, token traffic decreases, providing better utilization for heavily loaded networks.

The decrease in percentage of bus capacity consumed by propagation delays, shown by the dashed line with diamonds (blue), is due to two factors. First, as the number of data messages increases, the number of tokens decreases. Since messages are longer than tokens, there are consequently fewer total transmissions, and hence fewer propagation delays to suffer (one per message or token). Second, when a node transmits a data message it delays for a 2 microsecond interframe gap before it begins the next transmission, whether token or data message. When a node transmits a token no further transmissions can be made until the token is received by its successor and the successor begins transmitting.

3.2. Delay and Token Cycle Time

One of the most important characteristics of network performance is message delay. Delay (also called delivery time) spans the time between message arrival at the transmitting node and message arrival at the destination node and has three components. **Queueing delay** is the time from message arrival at a station until it reaches the head of the transmission queue; if there are no other messages enqueued at a station, then the queueing delay is zero. **Access delay** is the time between message arrival at the head of the transmission queue and the beginning of its transmission. **Transmission delay** is the time required for an entire message to be transmitted and to propagate to its destination. **Token cycle time** is the elapsed time from token arrival at a station until the token arrives again at that same station after transiting all other nodes in the logical ring.

Figure 2 illustrates the average delivery time and its two main components: queueing delay and access delay. Since message length is constant, transmission delay is constant and is not shown. As expected, the average delivery time grows exponentially as the offered load increases. Figure 2 shows that the average token cycle time also exhibits exponential growth as the offered load increases. At low offered loads token cycle time is short, equaling 64 token transmission times plus the time to transmit an occasional data message. As expected, token cycle time increases as offered load increases because there are more messages to send.

At low loads, access delay is the major contributor to delivery time as shown by the dotted line with circles (red); very few messages are simultaneously enqueued at a station so queueing delay, the dashed line with x's (green), is effectively zero. Simple analysis of Poisson arrival processes reveals that, because arrivals are exponentially distributed across the inter-service period, the mean time between arrival and service is one-half the inter-service interval. This is confirmed by comparing the access delay curve to the average token cycle time, the dot-dashed line with stars (blue). Messages wait an average of one-half a token cycle before they are transmitted, even in a very lightly loaded network.

As offered load increases the contribution of queueing delay increases until it becomes dominant at high loads. Queueing delay increases because there are more messages in the system as a whole, hence more messages ahead of an average message in the queue. Having more messages in the network increases the queueing delay because it increases the average token cycle time; having more messages in front of the average message further increases the queueing delay by the time needed to transmit the preceding messages.

Nevertheless access delay does not increase as dramatically as queueing delay because the increasing delay suffered by the messages at the head of the queue (while the token is elsewhere in the network) is balanced by the shorter access delays of all succeeding messages in the queue. The first message in the queue has an access time equal to a complete token cycle time (minus the time spent serving this station), while subsequent messages have an access delay equal to the transmission delay of the single preceding message.

4. Message Size

802.4 permits data frames of length from 12 to 8191 octets (8 bits each) including addresses and framing. Each such message is encapsulated by 96 bits (short addresses) or 160 bits (long addresses) of overhead (addresses, control, and framing). The data field can therefore vary from 0 to 8179 octets for short addresses or to 8171 octets for long addresses. In general, longer messages offer better performance than shorter messages because more data is carried relative to the frame overhead.

Figure 3 shows the variation in average delivery time as a function of message size. At low loads shorter messages offer better performance because the transmission time of larger messages is dominant; however, the average delivery times differ by a small amount. As the offered load increases, the average delivery times of the short messages increases much more rapidly than the average delivery times of the larger messages. Longer messages offer better performance at high loads because more data bits are transmitted per message, thereby reducing the framing overhead. Also, more data bits are transmitted before the interframe gap occurs.

5. Number of Stations

The performance of an 802.4 network is dramatically affected by the number of active stations. Performance declines (i.e., delivery time increases) as the number of active stations increases.

Increasing the number of active stations increases the number of token transmissions per token cycle. Thus messages on a large, lightly loaded network will have longer average delivery times than messages on a small, lightly loaded network. Concurrently, the increase in token transmissions decreases the bandwidth available for message transmission on heavily loaded networks.

Figure 4 shows how average delivery time is strongly influenced by the number of active stations. At low loads, average delivery time is proportional to the number of stations since each active station contributes a token transmission to the total token cycle time. At high loads, however, these additional tokens are consuming network bandwidth otherwise useful for carrying data, thereby magnifying the disparity between the delivery times of small and large networks. If delivery time must be bounded, then the number of active stations defines a practical limit on the total offered load which the network can carry.

6. Timers and Access Classes

802.4 offers the option of four *access_classes* (priorities). Timers associated with each *access_class* can ensure conformance to the priority scheme and can establish upper bounds on the token cycle time.

The *High_Priority_Token_Hold_Time* (*HPTHT*) establishes an upper bound on the time a station may serve its *Synchronous* traffic. When *Synchronous* traffic is exhausted or when the *token_hold_timer* expires, a station may conditionally serve its non-synchronous traffic. Service of the asynchronous classes is controlled by the value of three *Target_Rotation_Times:* TRT_{UA}, TRT_{NA}, and TRT_{TA} for *Urgent_Asynchronous*, *Normal_Asynchronous*, and *Time_Available* classes, respectively.

To properly enforce the priority option, the asynchronous timers should be set such that

$$TRT_{UA} > TRT_{NA} > TRT_{TA}.$$

Ordered this way, lower priority traffic is served only if there has been no higher priority traffic on this token cycle. If an *access_class* is to receive any service, its *TRT* must be greater than the token cycle time on an empty network.

For a network carrying asynchronous traffic, one can calculate the offered load at which the token cycle time equals the *TRT*. Let T_C be the token cycle time, X_m is the time to transmit one message (including propagation), X_t is the time to transmit one token (including propagation), N is the number of active stations, and λ is the (identical) message creation rate (in messages per second) at each station. Then,

$$T_C = N \cdot X_T + T_C \cdot N \cdot \lambda \cdot X_m$$

so

$$\lambda = \frac{T_C - N \cdot X_T}{T_C \cdot N \cdot X_m}.$$

Consider a network with a 10 Mbps bus and 64 stations, each emitting 256 bit messages and 96 bit tokens, with 10% of the load at the *Synchronous access_class* and 90% of the load at the *Urgent_Asynchronous access_class*. Token propagation time is 5 microseconds; messages are separated by a 2 microsecond interframe gap. With a TRT_{UA} of 2 milliseconds, the token cycle time T_C will equal TRT_{UA} at

$$\lambda = \frac{4ms - 64 \cdot \left(\frac{96}{10{,}000{,}000} + 5\mu s \right)}{4ms \cdot 64 \cdot \left(\frac{256}{10{,}000{,}000} + 2\mu s \right)}$$

so $\lambda \approx 430$ messages/second. With this message generation rate, total network offered load ρ is approximately 70 %.

Observation of Figure 5 shows that as the offered load increases the token cycle time also increases until it reaches the value of the *TRT*. The token cycle time does not increase beyond the *TRT* because the load from the *Synchronous access_class* is not large. At high loads, *Urgent_Asynchronous* traffic is delayed in favor of maintaining the token cycle time and transmitting the *Synchronous* traffic. We note that at the critical offered load point where the token cycle time equals the *TRT*, the delay experienced by the asynchronous messages increases three orders of magnitude!

Figure 6 illustrates the effect of the *TRT* on message throughput. As the offered load increases, message throughput for both *access_classes* increases, and the percentage of bus capacity consumed by token transmissions and propagation delays decreases. When the offered load increases past 70 percent of the bus capacity, the throughput of *Urgent_Asynchronous* messages, the line with squares, begins to decrease, and the percentages for token transmissions and propagation delays remain effectively constant. As the offered load increases further, the throughput of *Synchronous* messages continues to increase as the *Urgent_Asynchronous* message throughput decreases. In addition to the penalty of rapidly increasing delivery times seen as the offered load exceeds the limit set by the *TRT*, the restricted *access_class* also experiences a reduction in throughput; while larger number of messages arrive per second, fewer are transmitted.

7. Conclusions

Based upon our experimentation with the 802.4 protocol, we can make the following generalizations:

(1) Bus capacity is divided primarily among message throughput, tokens, and propagation delay. Message throughput increases linearly with offered load, ρ, until ρ nears bus capacity. The percentage of bus capacity consumed by tokens and propagation delays decreases linearly as ρ increases.

(2) As ρ increases, network access delay grows linearly while queueing delay grows

exponentially; thus total message delay grows exponentially with ρ.

(3) Short messages have shorter delivery times than long messages at low ρ; however, at high ρ, long messages are much more efficient users of network bandwidth and have much shorter delivery time than short messages. For a fixed message length, delivery times grows exponentially with ρ, but the rate of growth is strongly dependent upon the message length.

(4) Message delivery time is strongly affected by the number of active stations on the network. For any fixed offered load, the fewer the number of active stations the better the network performance (as characterized by shorter average delivery times).

(5) In networks with mixed priority traffic, message delivery time grows slowly with ρ until the token cycle time equals the asynchronous *TRT*; at that point the synchronous traffic delivery time is constant and maximized. The delay curve for asynchronous traffic likewise increases slowly with ρ until the token cycle time equals the *TRT*, but then increases several orders of magnitude over a very narrow range of ρ. Message throughput for the *access_class* is also reduced when the token cycle time exceeds the *TRT* for the *access_class* and as the offered load continues to increase.

References

[1] IEEE Std 802.4-1985, Local Area Network Standard- Token-Passing Bus Access Method and Physical Layer Specifications.

[2] IEEE Std 802.2-1985, Local Area Network Standard- Logical Link Control.

[3] Zimmerman, H., "OSI reference model - the ISO model of architecture for open systems interconnection," *IEEE Trans. on Communications*, April 1980.

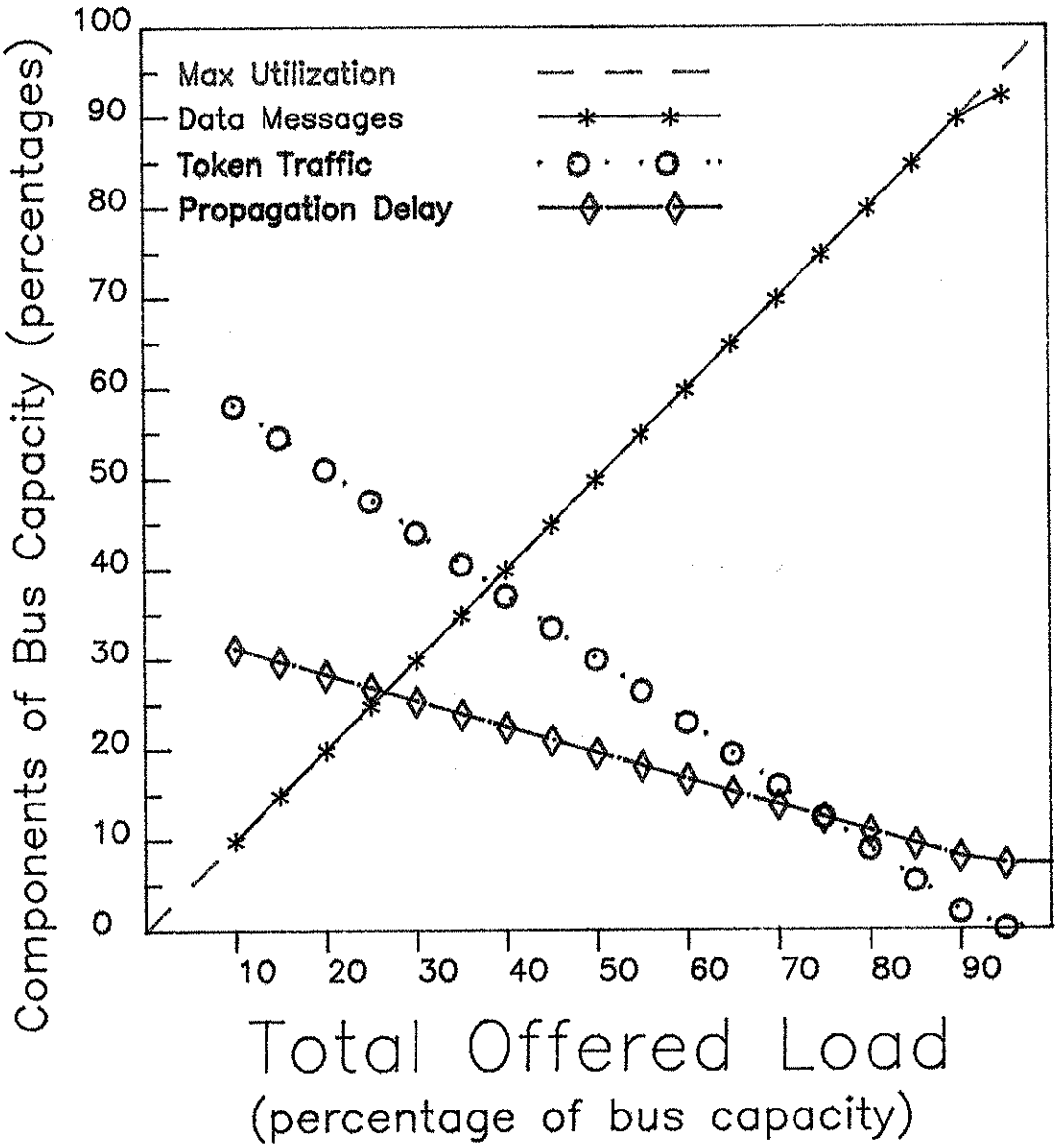

Figure 1 Base Configuration; Bus Utillization vs. Offered Load

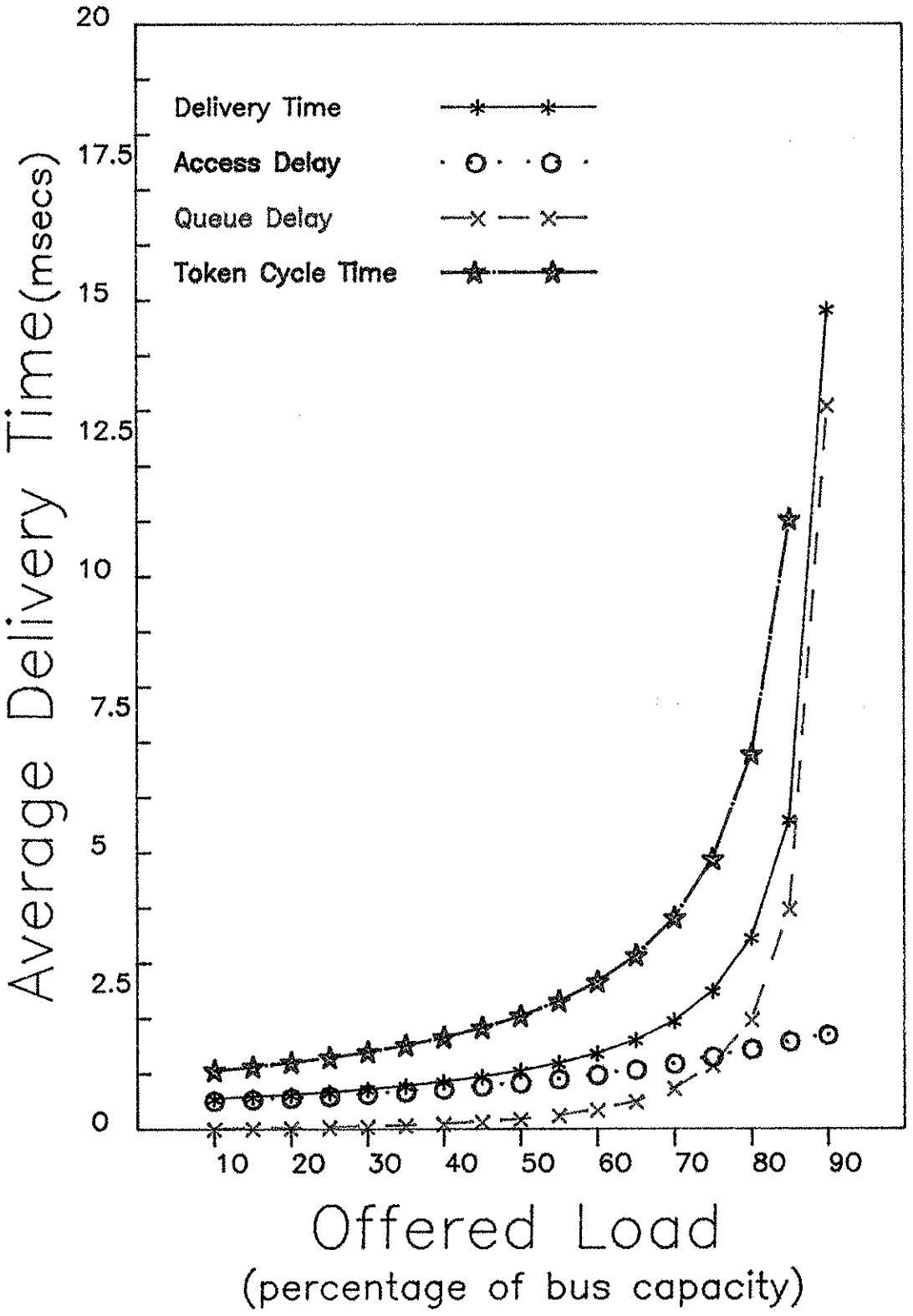

Figure 2 Base Configuration: Average Delivery Time, Its Components, and Average Token Cycle Time.

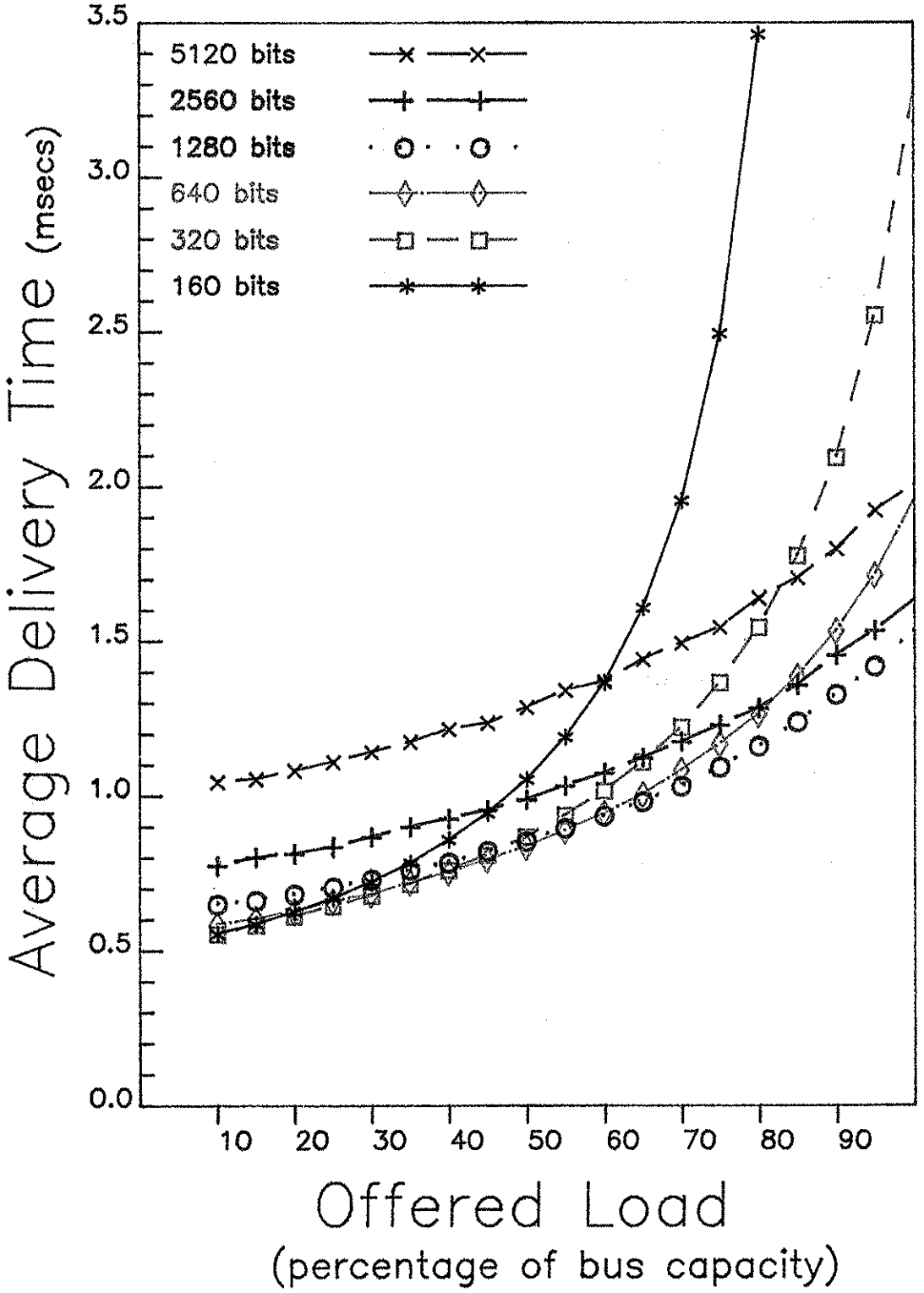

Figure 3 Average Delivery Time for Varying Packet Sizes

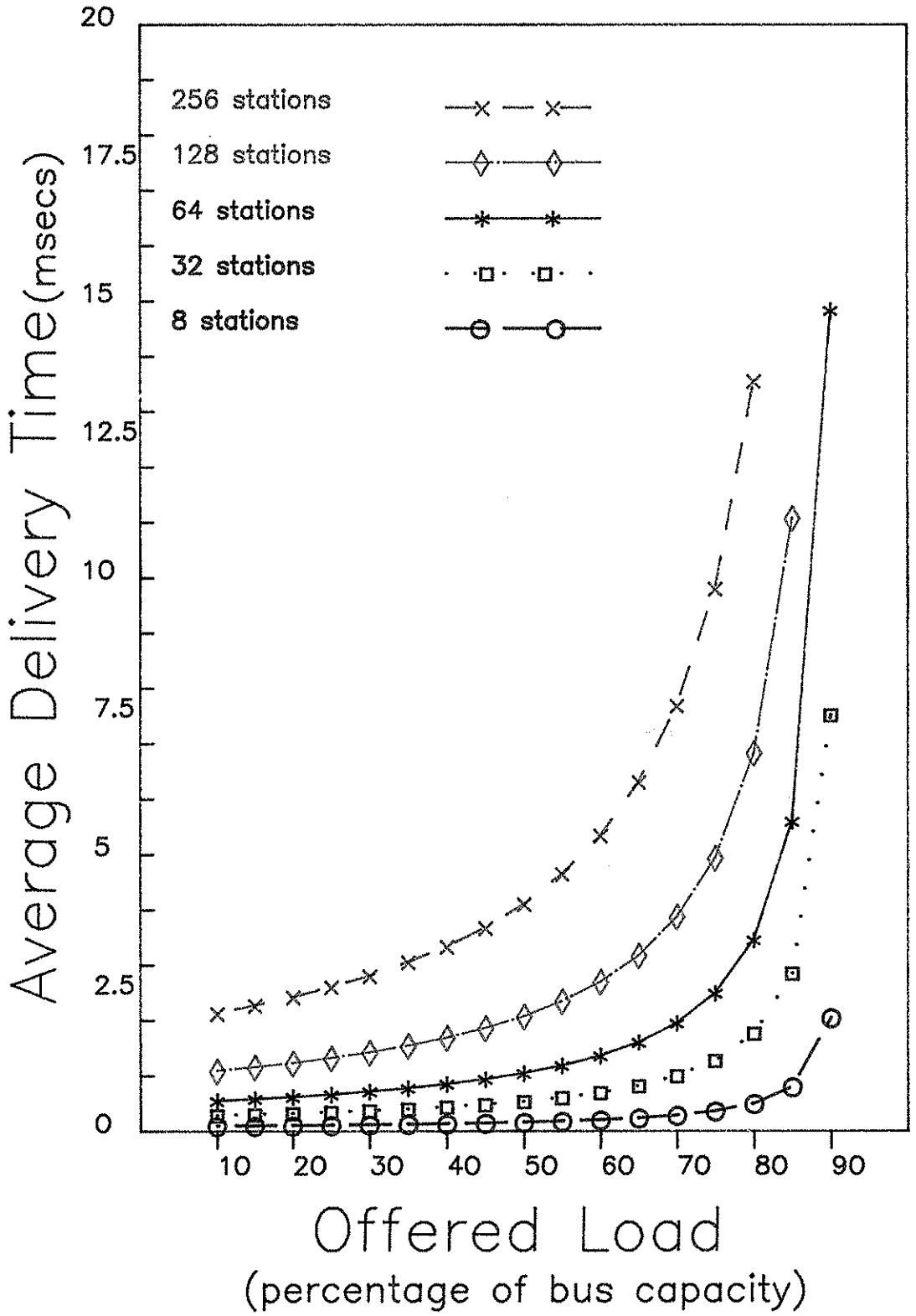

Figure 4 Average Delivery Time for Varying Number of Active Stations

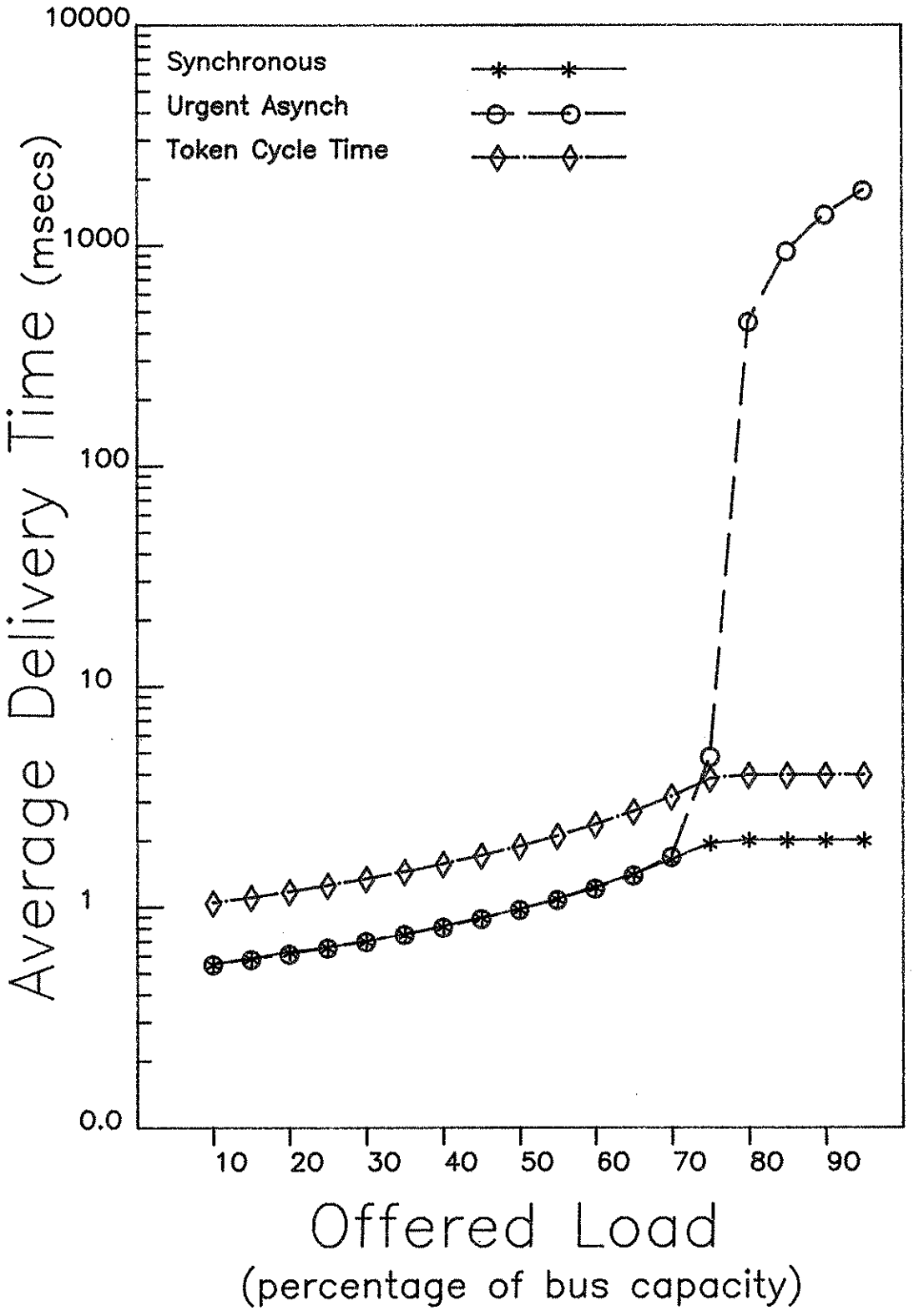

Figure 5 Average Delivery Time for Loads Distributed across the *Access_Classes* with a 4 msec *Target_Rotation_Time*.

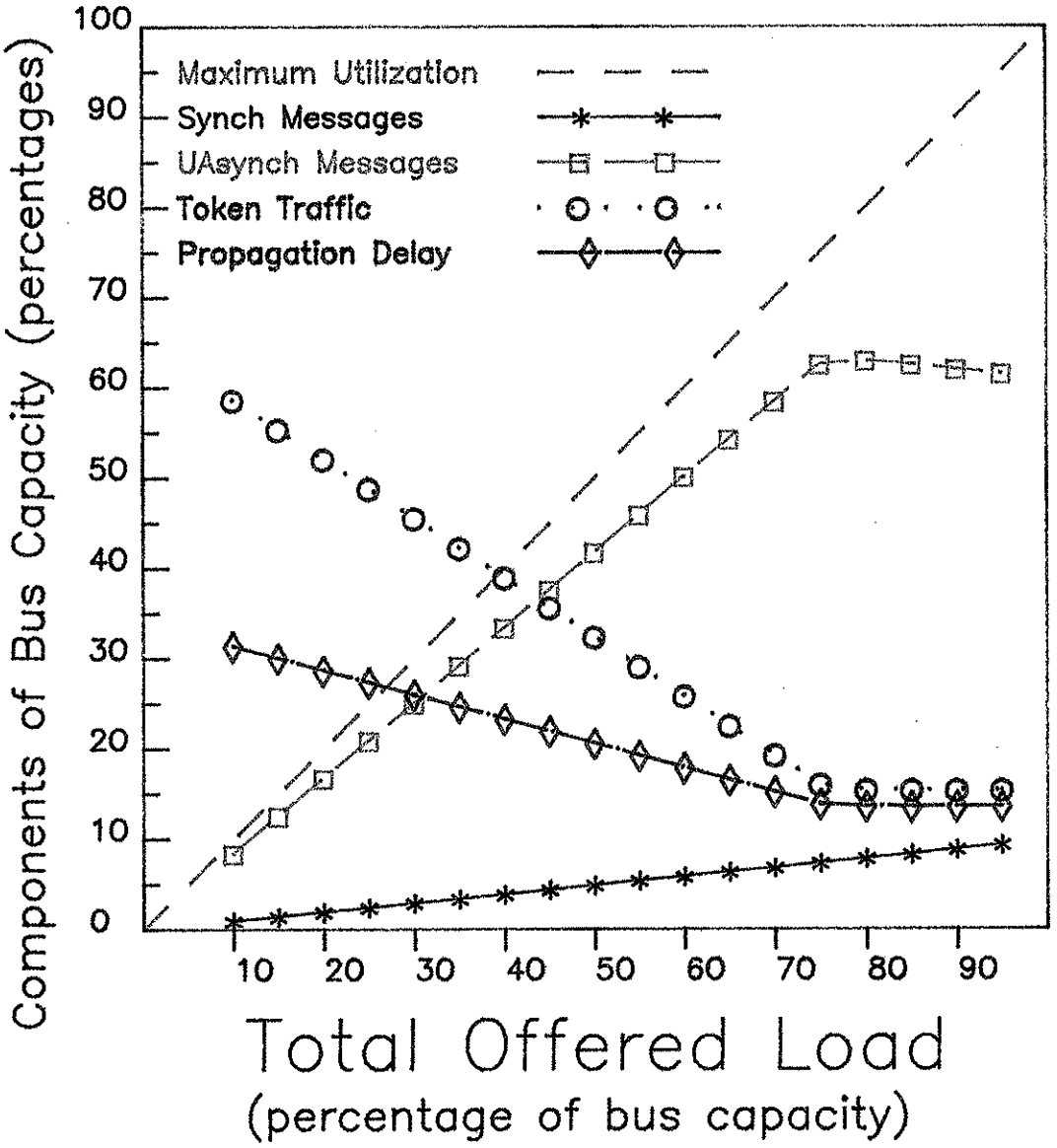

Figure 6 Components of Bus Utillization for Distributed Loads with a 4 msec *Target_Rotation_Time*