# The FDDI Protocol

Norman Matloff
University of California at Davis


November 30, 2001

## 1 Overview

One of the earliest types of local area networks was the **token ring**. As the name implies, the nodes are connected in a ring topology with point-to-point links. A special bit pattern called a **token** continually circles around the ring. Whenever a node wishes to send a message, it must wait for the token to get there. It then temporarily removes the token from the ring, sends its message, and then places the token back on the ring so that some other node may send.

The Fiber Distributed Data Interface (FDDI) is a newer type of token ring. It is fast, 100 Mbps, compared to the 4 Mbps of the IBM token ring and the original 10 Mbps rate for Ethernet. This speed was leapfrogged by Fast Ethernet at 100 Mbps, and now by Gigabit Ethernet at 1000 Mbps.

But FDDI has other advantages. First, it can be used on large networks, consisting of 500 nodes in a circumference of over 60 miles.[1] And FDDI is fault-tolerant (see the next section).

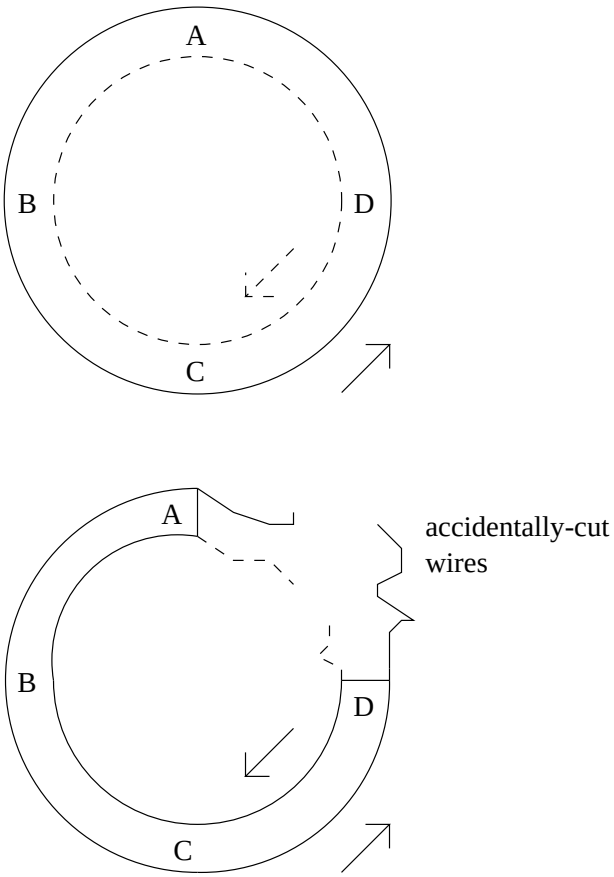
Due to the complexity of the FDDI protocol, FDDI interface chips are expensive, and not worth the cost for small networks. Thus FDDI's main usage has been for large **backbone** networks that connect together a number of smaller networks.

## 2 Fault Tolerance

FDDI is also built for fault tolerance. It is actually two networks, one going counter-clockwise and the other clockwise (as usually drawn). Normally only the primary links, i.e. the counter-clockwise ones, are used, but if a link or even a node goes down, the secondary links will maintain the operation of the network.

For example, consider a four-node network, with nodes A, B, C and D (in counter-clockwise order). Say the link from D to A is accidentally cut. Then the secondaries will automatically activate, so that the network makes "U-turns" at A and D: After a frame goes from B to C to D on the primary link, for instance, it will next go back to C from D on the secondary link, etc. In this manner, we retain the ring topology after the accidental break.

[1] These figures are for the optical-fiber version of FDDI, which is the common one. There are copper versions as well.




# 3 Capacity Allocation

After an FDDI node transmits its data, it will place a new token onto the ring, immediately following the last data frame it sends.[2] In ring network terminology, this is referred to as **early token release** or **release after transmit** (RAT), and contrasts to the other possibility, which is to not place a new token onto the network until the leading edge of the sender's frame circulates back to the sender. Clearly, RAT makes better use of network bandwidth.

FDDI distinguishes between **synchronous** and **asynchronous** traffic. The terms are misleading, as they do not have the same meaning as in other network contexts. Instead, the term "synchronous" simply refers to traffic which must be delivered in a timely manner, such as voice and video, and traffic which can tolerate delay, such as file transfers. What FDDI does then, is allow each node to transmit a given amount of synchronous traffic each time it gets the token, and transmit some asynchronous traffic if there is any "time left over," in a manner described below.

The heart of the FDDI approach is the setting of a Target Token Rotation Time (TRTT), which we will denote as $\tau_{target}$. Here the word "rotation" refers to the time between successive visits of a token to a given

[2]The term **token** here will correspond to the somewhat older term **free** token. In more recent literature, what was formerly called a **busy** token is now just considered part of a data frame, and thus not referred to as a token.

node, i.e. successive opportunities for a given node to transmit. Note that $\tau_{target}$ will be fixed, while the actual rotation time, $\tau_{actual}$, will vary, both from node to node and from cycle to cycle for the same node.

The word "target" is key, meaning that we aim for a token to rotate in $\tau_{target}$ time. Actually, we will see that the rotation time could be as long as $2\tau_{target}$.

Denote the nodes of the network by $N_i$, i = 0,1,2...,n-1. When a network is first brought up (or when a new node is added), the nodes negotiate a value for $\tau_{target}$. Each node $N_i$ bids a value of $\tau_{target}$ that is needed for its particular applications, so the final value becomes whatever the lowest bid turns out to be (so that all nodes have their timely-delivery needs satisfied). At that time, each node i will be assigned a value $\sigma_i$, which will be the amount of time the node will be allowed to send synchronous data in one turn, i.e. in one possession of the token. These amounts will satisfy the constraint

$$\sigma = \sum_{i=0}^{n-1} \sigma_i < \tau_{target} \tag{1}$$

This is a strict inequality (i.e. it uses $<$ rather than $\leq$), with the excess consisting of components such as the time that it takes one bit to traverse the ring. We will assume that it is the only component causing strict inequality above. Denoting the latency for a bit to travel from the $i^{th}$ to the $i+1^{st}$ node[3] by $\beta_i$, we then have

$$\tau_{target} = \sigma + \beta \tag{2}$$

where $\beta = \sum_i \beta_i$.[4]

When a node i receives a token, it can then send according to the following rules. First, the node calculates $\tau_{target} - \tau_{actual}$.[5] The token is then considered either "early" or "late," according to whether $\tau_{target} - \tau_{actual}$ is positive or negative, respectively. Then:

- Rule 1.

  Node i transmits as many synchronous frames as it has waiting for transmission, up to (but not exceeding) time $\sigma_i$.

- Rule 2.

  If the token had arrived early, node i is then allowed send as many asynchronous frames as it has waiting, for a total time of $\tau_{target} - \tau_{actual}$.[6]

Let us look at the implications of these rules. First, what is the worst-case value of $\tau_{actual}$? This will occur in the situation in which in one cycle no node sends anything, and in the next cycle each node sends the maximum allowed.

[3] Here and below, subscripts will be taken to use mod-n+1 arithmetic. Thus "$N_n$" is actually $N_0$.

[4] The token transmission time must also be accounted for, but we are assuming here that the token is so small that that time is negligible.

[5] Remember, the subtrahend here will vary from one node to another, and from one cycle to another.

[6] Plus "a little more," meaning that the final frame sent is allowed to cause the total asynchronous transmission time to exceed $\tau_{target} - \tau_{actual}$.

| token visit number | visit time and earliness, $N_0$ | visit time and earliness, $N_1$ |
|---|---|---|
| 1 | 0.0, NA | $\beta_0$, NA |
| 2 | $\beta$, $\sigma$ | $\beta + \sigma_0 + \sigma + \beta_0$, 0 |
| 3 | $2\tau_{target}$ , NA | NA, NA |

Specifically, look at any node, say $N_0$, and define time units so that current time is 0.0. Consider a cycle starting now, i.e. with the token leaving $N_0$ now, in which the token visits every node on the ring, but none of the nodes sends anything at all (not even synchronous data). After leaving $N_0$, the token will arrive at $N_1$ at time $\beta_0$, then arrive at $N_2$ at $\beta_0 + \beta_1$, and so on. (Here and below, refer to the table to more easily keep track of what happens at $N_0$ and $N_1$, for example; note that a zero value for earliness means a late visit.)

The token will make its next visit to $N_0$ at time $\beta$, i.e. $\tau_{actual} = \beta$. Thus the token will arrive to $N_0$ early, with the amount of earliness being $\tau_{target} - \beta = \sigma$.

Since the token arrived $\sigma$ amount of time early, $N_0$ is now allowed to send asynchronous data for $\sigma$ time, and it will do so, since we are assuming all the nodes are now on a sending binge. This means that it will send synchronous data for $\sigma_0$ amount of time, and then send asynchronous data for $\sigma$ amount of time. Then $N_0$ passes the token to $N_1$.

Now, at this point, what will be the value of $\tau_{actual}$ for $N_1$? The current time will be $\beta + \sigma_0 + \sigma + \beta_0$ (the token arrived at $N_0$ at time $\beta$, then $N_0$ sent data for time $\sigma_0 + \sigma$, then $N_0$ sent the token, which took time $\beta_0$ to reach $N_1$). The time when the token last reached $N_1$ was $\beta_0$. So, $\tau_{actual}$ would then be

$$\beta + \sigma_0 + \sigma = \tau_{target} + \sigma_0 \tag{3}$$

In other words, the token arrives at $N_1$ late! Thus $N_1$ is not allowed to send any asynchronous data. But it is allowed to send synchronous data, for $\sigma_1$ amount of time, and will do so.

Continuing this reasoning, you can see that $N_2$ will now not be allowed to send any asynchronous data, but it will still send its synchronous data for $\sigma_2$ amount of time, and so on. And then, adding up the results of the actions at all the succeeding nodes, we can see that the token will make its next vist to $N_0$ at time

$$\beta + \sigma + \sum_i (\sigma_i + \beta_i) = 2\tau_{target} \tag{4}$$

Since the previous visit to $N_0$ had been at time $\beta$, that means

$$\tau_{actual} = 2\tau_{target} - \beta \leq 2\tau_{target} \tag{5}$$

So an upper bound for $\tau_{actual}$ is $2\tau_{target}$. (And if $\beta$ is small relative to $\sigma$, the inequality above will essentially be an equality.)

In other words, each node is secure in the knowledge that it will need to wait no longer than $2\tau_{target}$ to have a chance to send.[7]

Moreover, in the scenario outlined above, all nodes will find that the next time the token pays them a visit, it will be not early. That means that they will not be allowed to send any asynchronous frames in this next

[7] On the other hand, keep in mind that the queue within a node could be quite long, and thus any given frame may have to wait much longer than this to get out onto the network.

turn, so that the scenario could not repeat itself in consecutive turns.

We can also see that the value of $\tau_{actual}$ at one node will have a direct impact on the value at the node's immediate downstream neighbor. For example, using the following argument we can see that if there is no synchronous traffic, then $\tau_{actual}$ will be at most $\tau_{target} + \phi$, where $\phi$ is an assumed upper bound on the amount of asynchronous traffic a node will ever have available to send at a given time.

Here is how that comes about: Consider consecutive nodes A and B, and look at two consecutive rotations of the token. Say the token visits A at time $t_{A1}$, whereupon A transmits asynchronous data for time $X_{A1}$, after which it sends the token to B. Denote the propagation delay between A and B by $\alpha$.

Then the token arrives at B at time

$$t_{B1} = t_{A1} + X_{A1} + \alpha \tag{6}$$

B then sends asynchronous data for time $X_{B1}$. Later, at time $t_{A2}$, the token makes its next visit to A; A sends for time $X_{A2}$, and then sends the token to B, arriving at time

$$t_{B2} = t_{A2} + X_{A2} + \alpha \tag{7}$$

Let $\tau_{actual,A}$ be the target rotation time for A in this situation, i.e.

$$\tau_{actual,A} = t_{A2} - t_{A1} \tag{8}$$

and define $\tau_{actual,B}$ similarly for B:

$$\tau_{actual,B} = t_{B2} - t_{B1} \tag{9}$$

Now, remember, we are trying to show that

$$\tau_{actual} \leq \tau_{target} + \phi \tag{10}$$

for all nodes if there is no synchronous traffic. With this goal in mind, suppose that this inequality held for node A in the situation outlined above, i.e.

$$\tau_{actual,A} \leq \tau_{target} + \phi \tag{11}$$

Then what about $\tau_{actual,B}$? Well, by performing some algebra on Equations (6), (7), (8) and (9), we would have

$$\tau_{actual,B} = \tau_{actual,A} + X_{A2} - X_{A1} \tag{12}$$

Now consider two cases, the first one being that $\tau_{actual,A}$ is greater than $\tau_{target}$. In this case, the token's second arrival to A is late, so A is not allowed to send any asynchronous data, i.e. $X_{A2} = 0$. Thus, Equation (12) implies that

$$\tau_{actual,B} = \tau_{actual,A} - X_{A1} \leq \tau_{actual,A} \tag{13}$$

with the inequality coming from the fact that $X_{A1} \geq 0$. Thus Equation (13) shows that if Equation (10) holds for A then it holds for B too.

In the second case, where $\tau_{actual,A}$ is less than $\tau_{target}$, then from Equation (12)

$$\tau_{actual,B} = \tau_{actual,A} + X_{A2} - X_{A1} \leq \tau_{target} + X_{A2} - X_{A1} \leq \tau_{target} + \phi \tag{14}$$

where the last inequality is due to the fact that each of $X_{A2}$ and $X_{A1}$ is between 0 and $\phi$ and thus their difference is no more than $\phi$. So, in this case again we have that 10 holds for B if it does for A.

We are almost done, except that we must deal with the word "if" in the last sentence. All we have shown so far is that once one node satisfies (10), the next node will, and thus so will the next node after that, and so on. But when the network first comes up and no node has anything to send, the first rotation of the token, taking time $\beta$, will indeed take less than $\tau_{target}$ (see Equation (2)), so that starts the ball rolling. The argument above shows that $\tau_{actual}$ will continue to be less than $\tau_{target} + \phi$ after that, for all nodes and all cycles of the token around the ring, even though $\tau_{actual}$ will vary within the ranger below this value.