The FDDI MAC Meets Self-Stabilization

Adam M. Costello
University of California at Berkeley
http://www.cs.berkeley.edu/~amc/

George Varghese
Washington University in St. Louis
http://www.ccrc.wustl.edu/~varghese/

Abstract

A self-stabilizing protocol is one that can continue to operate correctly after arbitrary corruption of protocol state variables and the injection of arbitrary messages. Self-stabilizing protocols are extremely robust and can be simpler than other protocols (because they use a uniform mechanism to deal with a large number of possible faults). We demonstrate these advantages by modifying the Fiber Distributed Data Interface Media Access Control (FDDI MAC), a widely used token-ring protocol, to make it self-stabilizing. Our modifications are simple: the major step is to add a token rotation counter and a phase bit that are carried in all messages. Not only is our modified protocol more robust (for example, our protocol recovers from multiple tokens in less than 5.75 ms, while the existing MAC might never recover), but it also recovers from lost tokens more quickly (0–0.36 ms versus 2.5–4.1 ms). We describe the modest changes made to the existing MAC state machine.

1 Introduction

Self-stabilizing protocols were introduced by Dijkstra [Dij74]. A protocol is *self-stabilizing* if, when started from an arbitrary global state, it exhibits "correct" behavior in bounded time. Typical protocols cope with a specified set of failure modes such as packet loss and link failures. A self-stabilizing protocol copes with a set of failures that subsumes most previous categories, and is robust against transient errors, including memory corruption and the injection of corrupt packets. Transient errors do occur in real networks and cause systems to fail unpredictably [Ros81, Per83]. Thus stabilizing protocols are more *robust* than traditional protocols. Furthermore, they are *simpler*, because they use uniform mechanisms to deal with different failures and need no special mechanisms for initialization.

In this paper we describe a self-stabilizing version of the FDDI MAC [ANS87] to demonstrate the feasibility of self-stabilization in real protocols. The paper is organized as follows: Section 2 describes a theoretical self-stabilizing token-passing protocol. Section 3 reviews FDDI, and Section 4 identifies new requirements that arise when one tries to apply the theoretical protocol to a real-world token ring. Section 5 presents modifications to the FDDI MAC protocol to incorporate self-stabilization while satisfying the new requirements. Section 6 proves that the new protocol stabilizes. Sections 7 and 8 examine the costs and benefits of the modified protocol. Section 9 gives some simulation results, and Section 10 states our conclusions.

2 Theoretical Protocol

Here we review a self-stabilizing token-passing protocol that uses a technique called *counter flushing* [Var94], based on ideas from [Dij74]. The abstract ideas are:

- Nodes periodically inform their neighbors of "where" in the execution they are.
- There is a unique leader node who is always "right" by definition.
- Whenever a discrepancy is discovered, the leader ignores it, but a non-leader will re-sync by altering its own state.

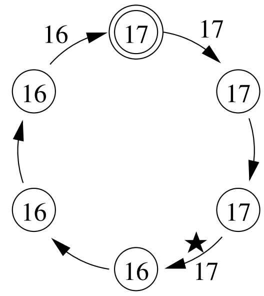
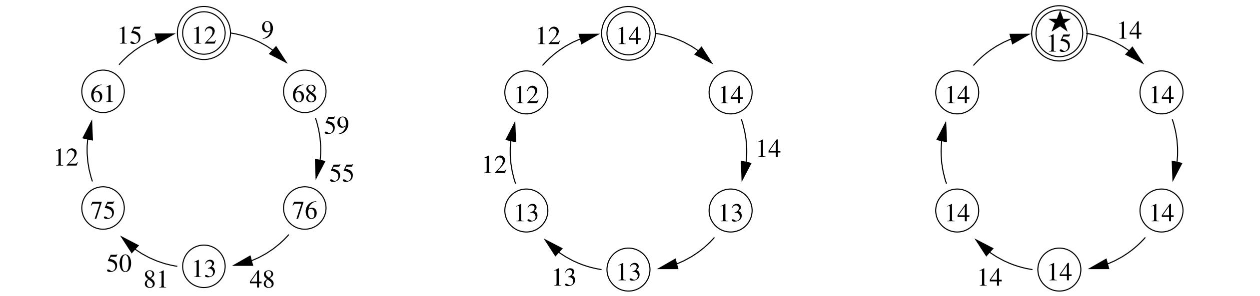


Figure 1. A valid state of a token ring using counter flushing.



In the initial state, the ring is full of garbage.

After one round-trip delay, all garbage values have been flushed out.

The leader is stuck until its value has filled the ring (at most one more round-trip).

Figure 2. Stabilization of the token ring using counter flushing.

The protocol works as follows. Number the nodes sequentially clockwise, starting with 0 at the leader. As in a normal ring, a token is sent (clockwise) around the ring such that at most one node has the token at a time. For fault-tolerance, let each token packet carry a counter value c indicating how many times the token has circumnavigated the ring, and let every node i store the counter value c_i of the last token that arrived at the node. To guard against lost tokens, each node i periodically retransmits a token packet containing c_i. Nodes must use their counters to weed out duplicate tokens. In a valid state, there are only two values in the ring: one arc, starting with the leader and continuing partway around the ring, contains only the value c_0, while the rest of the ring contains only the value $c_0 - 1$. The token is at the end of the first arc. For example, Figure 1 shows a valid state with the token in transit from node 2 to node 3.

When a non-leader node i receives a token packet from upstream, it compares the counter value in the packet (c) with its own counter value (c_i). If they are *different*, the node assumes it has received the token and sets c_i equal to c, but if they are *equal*, the node ignores the token packet. The leader, however, follows a different rule: if the counter value c in the packet is *equal* to the leader's counter value c_0, the leader assumes it has received the token and increments its counter value modulo M, but if c is *different* from c_0, the leader ignores the token packet. For example, from the state shown in Figure 1, the value 17 will sweep around the ring as the token moves clockwise. When the token returns to the leader, the leader increments to 18 and begins a new cycle.

In the initial state, the counter values at nodes and in tokens stored on links may be arbitrary. Let c_{max} be the maximum number of counters that can be stored in the ring in the initial state. If the modulus M is greater than c_{max}, then the protocol stabilizes within two round-trip delays, where a round-trip delay is the time for a causal chain of packets to propagate all the way around the ring. Once the first leader-produced value has made its way all the way around the ring and is about to arrive back at the leader, the initial garbage values have all been flushed out, and only values produced by the leader are left in the ring. Since the leader could have incremented at most once for each distinct value initially in the ring, and $M > c_{max}$, the leader's counter value cannot appear anywhere except in a single arc starting at the leader. Therefore, the leader will not increment again until its current value has filled the ring, which takes at most one more round-trip delay, at which point the ring is in a valid state. Figure 2 shows an example stabilization scenario.

3 Overview of FDDI

3.1 Normal Operation

Fiber Distributed Data Interface (FDDI) is a local area network standard using a token ring architecture. Up to 1000 stations are connected in a ring by up to 200 km of glass fiber. Data is transmitted at a rate of 25 million *Symbols* per second. There are sixteen data symbols (0–9, A–F), each representing a group of 4 bits, and 8 control symbols (H, I, J, K, Q, R, S, T). The data rate is therefore 100 Mbps.

Data bytes are grouped into *Frames* with headers and trailers, which contain delimiters, message type information, source and destination addresses, the payload, a checksum, and status flags. Only the station holding the token may send Frames—the other stations repeat the Frames, and any station (particularly the one matching the destination address) may copy the Frame and deliver the data up to a higher layer. When the token-holder is done transmitting, it relinquishes the token by sending a short *Token* message containing only the delimiters and message type. Another station may *capture* the token by deliberately failing to repeat the entire Token message.

FDDI provides latency bounds using *timed token access*. There is a *Target Token Rotation Time* (TTRT) agreed upon by all stations in the ring, and the stations cooperate to insure that the token circulates at least once every TTRT. There are two classes of Frames—synchronous and asynchronous.

Each station may have a *synchronous allocation*, which is a percentage of the TTRT (higher layers insure that the sum does not exceed 100%). Whenever a token arrives, a station may capture it and send synchronous Frames for its allocated time. After that, it may send asynchronous Frames, but only as long as the time since the previous token arrival is less than TTRT. If TTRT was already exceeded after the synchronous Frames were sent, no asynchronous Frames may be sent this time around. In this way, latency-sensitive (synchronous) traffic takes priority over other traffic. The actual token rotation time is provably TTRT on average, and $2 \times \mathrm{TTRT}$ at most.

To prevent Frames from circulating around the ring multiple times, causing duplicate deliveries at the receiving station, Frames are *stripped* by their senders. Every station, while repeating a Frame, inspects the source address. If it sees its own address, the station stops repeating, and sends Idle Symbols (I) instead. The result is a *fragment* (partial message) that circulates harmlessly around the ring until it encounters a transmitting station, which does not repeat it.

3.2 Fault Recovery

Each station has a Valid Transmission Timer (TVX) and a Late Counter (Late_Ct) for detecting faults. TVX is reset to -2.5 ms each time a valid Frame is received. If it ever counts up to 0, the station will reset the ring (see below). Late_Ct counts multiples of TTRT since the token last arrived at the station. If it reaches 2, something must be wrong, so the station will reset the ring.

A station resets the ring by continually issuing *Claim Frames* containing the station's desired TTRT. Any station, upon receiving a Claim Frame from another station, repeats it, or, if its own desired TTRT is smaller, begins issuing its own Claim Frames (source addresses are used as tie-breakers). Eventually, the station with the smallest bid receives its own Claim Frame, knows that it has won the bidding, and issues a Token. This reset mechanism is called the *Claim Token process*.

If a station sends Claim Frames for a long time without receiving a lower bid or its own Claim Frame, it initiates the *Beacon process* to locate the presumed break in the ring. It continually sends *Beacon Frames*. Any station, upon receiving a Beacon Frame from another station, repeats it. Therefore, only the station immediately downstream of the break will continue to send Beacon Frames, a fact useful for locating the break. If a station receives its own Beacon Frame, it knows that the physical ring has been restored, and resets the ring using the Claim Token process.

A lost token will always be recovered. Usually, its absence will be detected by TVX, because if no station has the token, no station may create any Frames, and existing Frames should get stripped. But if a Frame exists with a source address matching none of the stations on the ring (perhaps caused by line noise) and the token is lost, then the Frame will circulate repeatedly, preventing any station's TVX from expiring. The Late Counter, however, will catch the fault, and cause the ring to be reset.

The problem of multiple tokens, on the other hand, might never be corrected. This situation might arise, for example, if a Token fragment is corrupted by line noise back into a Token. A Token fragment is a Token with its last three symbols missing, the result of a Token being captured by a station. Once there are two tokens in the ring, both will persist until a Token message encounters a token-holder, but there is no bound on how long that could take. Both tokens could propagate around the ring at equal average rates. Meanwhile, any data Frame sent by one token-holder to a station on the far side of the other token-holder will not reach its destination, because the second token-holder will not repeat it. Because Tokens and Frames are circulating, TVX and the Late Counter will not detect a problem.

4 Concerns of Real Token Rings

The counter flushing protocol described in Section 2 requires all nodes, even those lacking a token, to send their counter values periodically. In a real ring, this raises the problem of what to do if a frame arrives while the counter value is being sent. Enqueuing the incoming frame would add latency at each station in the ring, which is unacceptable. FDDI goes out of its way to make sure that there is no need for more than a single symbol's worth of latency at each node, because a ring may contain up to 1000 stations. Another option, interrupting the outgoing counter, would make it difficult to insure that the counter ever gets sent—perhaps every attempt will be interrupted.

The solution we use is to include a counter value in every frame. Each non-leader sends its counter value once when it gives up the token, but does not periodically resend it. Instead, only the leader resends its counter value, and non-leaders let those values pass through. In order for a node to capture a token, it must prevent the counter value from reaching nodes downstream from itself. It could do this either by replacing the passing counter value with its old counter value, or by clearing a special bit in the message.

This choice highlights a second new concern: the distinction between the ring latency and the token rotation time. In the theoretical protocol, they were one and the same, but in a real ring, the token rotation time can be *much* greater than the ring latency. Therefore, we must allow counter values to pass through a station holding a token, so we add a special bit to the messages that the token-holder can clear to prevent the message from carrying a token to the next station. When the bit is set, the message might or might not carry a token, but when the bit is unset, it certainly does not.

5 Protocol Changes

This section describes the modified FDDI MAC protocol. For a detailed list of changes to the specification [ANS87] see http://www.cs.berkeley.edu/~amc/research/fddi/.

5.1 Leader Selection

The theoretical counter flushing protocol assumes the existence of a unique leader node. In a real token ring, we must elect the leader, which means we need a self-stabilizing leader election algorithm. Such algorithms exist, and we believe we could adapt them to the FDDI MAC; however, we have not yet done this.

We have set aside the leader election problem as an orthogonal concern, and for the time being we use the FDDI MAC's existing Claim Token process as a non-stabilizing leader election algorithm. In the modified protocol, the winner of that process not only creates the initial token, but also becomes the Leader. Because the Claim Token process is non-stabilizing, the modified protocol does not necessarily recover from conditions that violate the model's assumption of a unique leader node. For example, a crashed Leader can leave the ring with zero Leaders and multiple tokens; or memory corruption or line noise could cause more than one station to believe it had won the Claim Token process, resulting in a ring with multiple Leaders. Replacing the Claim Token process with a self-stabilizing leader election algorithm would solve this shortcoming.

5.2 Token Class Simplification

We next remove support for the *restricted* class of tokens. According to [Jai94], restricted mode "is not known to be used by anyone". Supporting it would have been an uninteresting complication.

5.3 Message Format

Next we modify the message format, as shown in Figure 3. In an FDDI ring, every valid message is either a Token or a Frame, and is composed of Symbols. Both Tokens and Frames begin with a Start Delimiter (2 Symbols), followed by a Frame Control sequence (2 Symbols), which identifies the message type (Token, data Frame, Claim Frame, etc.). A Token has an End Delimiter immediately following the Frame Control, while a Frame has several fields that do not concern us. In both Tokens and Frames, immediately following the Frame Control, we add a new Rotation Counter field (5 Symbols), which encodes a 12-bit counter in 20 bits for error detection. Then comes the new Phase Bit field (1 Symbol), which encodes the aforementioned "special bit" in

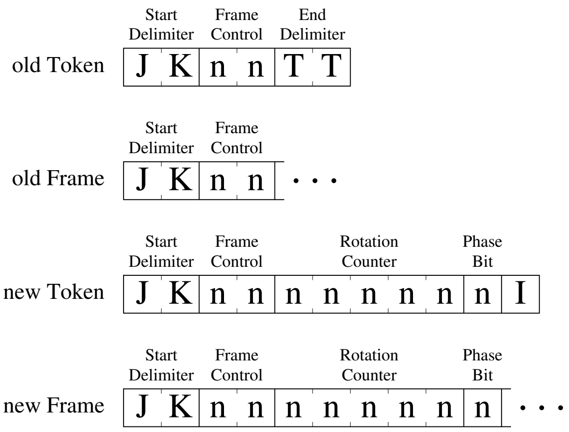


Figure 3. Comparison of old and new Token (Tframe) and Frame formats.

4 bits for error detection. These additions reduce the maximum payload from 4478 bytes to 4475 bytes. In Tokens we also replace the End Delimiter[1] by an Idle Symbol, so that a Token looks like a fragment. Tokens and Frames will be treated identically by receivers. The Rotation Counter and Phase Bit together take the place of the single "counter value" of the theoretical technique.

Now a station can acquire the token not only via a Token, but also via a Frame or fragment, though that happens only during stabilization. Even during normal operation, a Token does not necessarily carry a token—it just carries a Rotation Counter and Phase Bit, which together may or may not represent a token. A message with an unset Phase Bit never carries a token. To avoid confusion, Tokens that do not necessarily carry tokens will henceforth be called Tframes.

5.4 Station Behavior

Each station contains an internal register with a rotation counter and a phase bit[2]. Any station, when creating a Tframe, copies its internal rotation counter and phase bit into the Tframe. (Non-Leaders create Tframes under the same circumstances as in the old protocol—whenever they finish transmitting. The Leader additionally creates Tframes whenever it is idle.) Any station, when creating a Frame, copies its internal rotation counter into the Frame, and clears the Frame's Phase Bit regardless of its internal phase bit, so that it cannot give away the token while it is transmitting.

[1] The Token End Delimiter was formerly used to confirm the validity of a Token, but in the modified protocol, that determination must be made as soon as the Phase Bit is received, before it is repeated.

[2] Lower case is used here because the internal register does not necessarily use the same encoding as the Rotation Counter and Phase Bit fields of Frames and Tframes.

```
Receiver
while true do
    wait until start of message
    send start-of-message signal to transmitter
    if Rotation Counter and Phase Bit are error-free then
        if I am Leader then
            if Rotation Counter and Phase Bit equal
            internal register then
                increment internal rotation counter
                signal transmitter: token-received
        else
            copy them into internal register
                if register changes and Phase Bit is set then
                    signal transmitter: token-received
    wait until end of message or Source Address is my own
    signal transmitter: end-of-message

Transmitter
while true do
    repeat
        if I am Leader then
            whenever queue is empty, send Tframe carrying
                internal Rotation Counter and a set Phase Bit
        else
            send Idle symbol
    until start-of-message signal
    repeat message until any signal (if Leader, overwrite
    Rotation Counter and Phase Bit from internal register)
    if token-received signal and token is usable then
        if I am Leader then send a set Phase Bit
        else send an unset Phase Bit
        while I am allowed to hold token
            send Frames with unset Phase Bits, copying
            Rotation Counter from internal register
            break on token-received signal
            or if internal phase bit is unset
        send Tframe, copying Rotation Counter and Phase Bit
        from internal register
```

Figure 4. Pseudocode for station behavior. The receiver and transmitter run in parallel, and register read/writes are atomic.

Non-Leader Behavior: Frame repetition is handled differently by the Leader and non-Leaders. Non-Leaders copy all incoming Rotation Counter and Phase Bit fields into their registers (if no errors are detected in either). If this changes the register, and if the phase bit is now set, then the message carries a token, which the station may optionally capture by clearing the outgoing Phase Bit. As in the old protocol, non-Leaders need only a single Symbol of buffer space.

Leader Behavior: The Leader, on the other hand, copies in the other direction—from the internal register into the Rotation Counter and Phase Bit of the message. It then compares the original received Rotation Counter and Phase Bit with its internal values. If they were equal, the message carried a token, and the Leader increments its internal rotation counter. The Leader cannot help but capture the token, but can release it just as effortlessly. The Leader's internal phase bit must *always* be set[3], even during stabilization, so it actually uses the constant 1 rather than a register.

The Leader transmits Tframes rather than Idle Symbols when it has nothing else to send or repeat (hence the effortless release). This requires it to transmit into a queue rather than directly onto the medium. The queue must be large enough to hold one Tframe (27 Symbols, including the required 16 Idle Symbols of Preamble[4]). Symbols are dequeued onto the medium. Whenever the queue becomes empty, it is loaded with a Tframe. Thus, the Leader keeps the ring filled with messages. The largest gaps occur when Frames are stripped by non-Leaders, but the size of a gap cannot exceed the maximum Frame size (9000 Symbols).

The subset of station behavior pertinent to this section is shown as pseudocode in Figure 4.

5.5 Ring Operation

All remaining aspects of the original protocol remain unchanged, particularly rules regarding when tokens may be captured, how long they may be held, priorities, frame stripping, resets, statistics counters, and Beacons.

In normal operation, there will be no more than two rotation counter values in the ring, one stretching from the Leader to some point, and the previous one in the remainder of the ring. Also, phase bits will be set between the Leader and the token holder (who has the same rotation counter as the Leader), unset between the token holder and the boundary between the new and old rotation counter values, and set in the remainder of the ring. The token holder will send Frames with unset Phase Bits for a while, then send a Tframe with a set Phase Bit. This Phase Bit will be copied into successive stations until it arrives at one willing to capture the token. This process is repeated until the set Phase Bit makes its way back to the Leader, which increments its rotation counter, starting the whole cycle over again. The new rotation counter gets copied into all the stations in about one ring latency, even though it typically takes the the token much longer to traverse the ring. Therefore the third arc, containing old rotation counter values and set phase bits, is usually empty. See Figure 5.

In correct operation, a station holding a token should never receive a token, nor should its phase bit be 0. Therefore we require a token holder to issue a Tframe immediately if either of these conditions is violated. While not strictly necessary, this policy speeds up the stabilization process.

[3]Conceptually, a set internal phase bit means "I have seen the token this round". Because each new round begins when the token returns to the Leader, the Leader's phase bit is always set.

[4]It *may* be necessary for the Leader to follow the token with 15 extra Idle Symbols, to provide a full Preamble for the next repeated Frame, in which case the buffer must hold 42 Symbols.

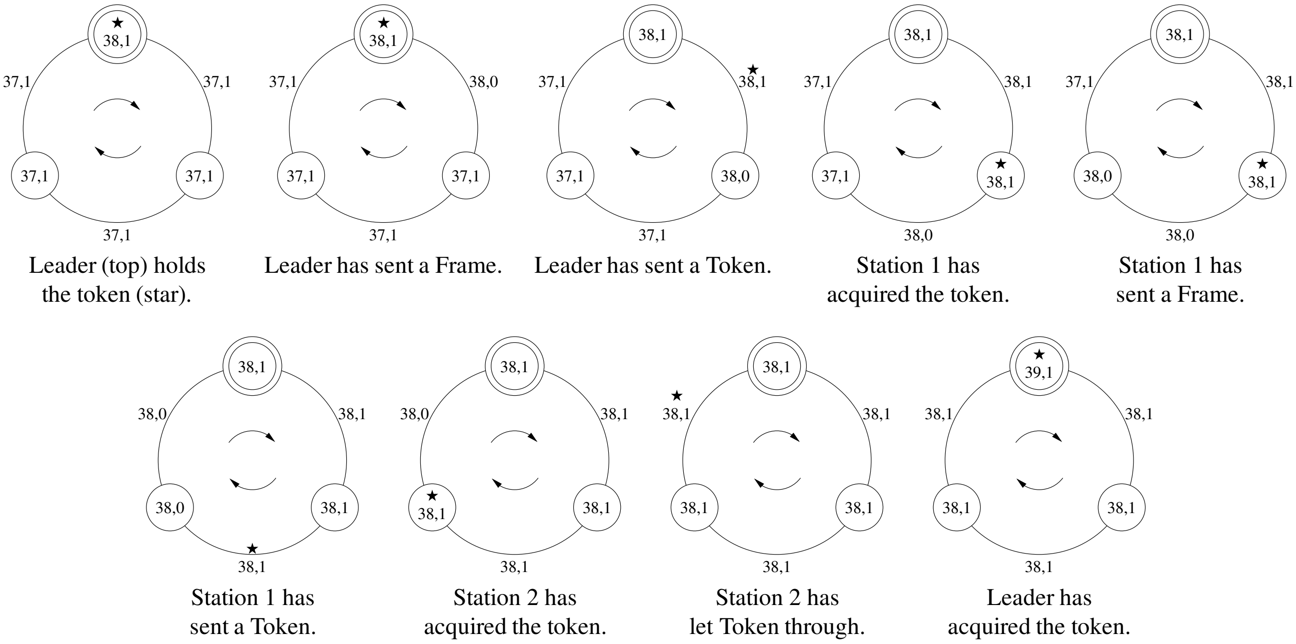


Figure 5. Example operation of the modified FDDI ring, shown after each 1/6 rotation. The rotation counter propagates quickly, while the token more slowly follows the phase bit boundary.

6 Stabilization

A walk-through of the stabilization is shown in Figure 6. It is similar to the stabilization of the theoretical token ring shown in Figure 2. A formal proof of stabilization follows.

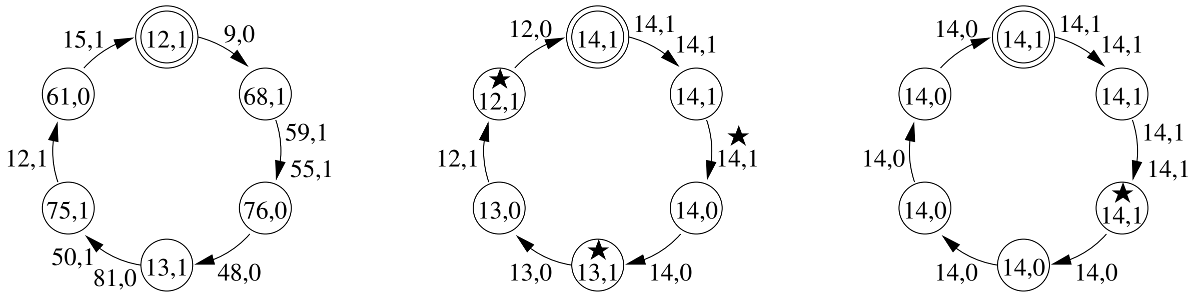
Start the protocol at time 0 in a state with arbitrary rotation counters and phase bits in the stations, an arbitrary number of stations that think they have a token, and arbitrary Symbols on the ring. We do insist, however, that there be exactly one Leader. If we were using a self-stabilizing Leader election algorithm, that would be guaranteed to become true within a bounded time.

Let n be the number of stations in the ring (at most 1000), let L be the total propagation delay (latency) around the ring (at most 1.617 ms), let TT be the time to transmit a Tframe (1.08 μs), and let $FT_{\max}$ be the time to transmit a maximal Frame (0.36 ms).

At time 0, the Leader may be in the middle of sending a Frame, but within $FT_{\max}$ it will send another message carrying its rotation counter. Thereafter, the Leader regularly emits messages carrying its rotation counter. It includes them in all messages it creates, overwrites them in all messages it repeats, and fills gaps with new Tframes. Imagine that rotation counters (residing in messages or in stations) initially in the ring at time 0 are colored red, except the Leader's, which is green. As time passes, rotation counters that are "descended" from green rotation counters are also green. A green arc will steadily grow, starting at the Leader and proceeding around the ring. When the rotation counter at the frontier of the arc encounters a station, it is copied into the station's internal rotation counter, which turns it green. Then one of the following things happens:

- If the station did not have a token, it repeats the message (possibly clearing the Phase Bit), so the green arc continues to grow with no delay at this station.
- If the station had a token, and the incoming message caused it to issue a Tframe, then the green arc continues to grow, with a delay of TT at this station.
- If the station had a token, and the incoming message did not cause it to issue a Tframe, then the incoming rotation counter must have matched the internal rotation counter, which in turn matches the Rotation Counter in the currently outgoing Frame[5], so we might as well consider that Rotation Counter to be green. Hence the green arc continues to grow. The delay at this station might be as much as TT if the outgoing Frame had just barely begun when the incoming rotation counter arrived, or might be less than zero if the outgoing frame was already well underway when the rotation counter arrived.

[5] That is not necessarily true between time 0 and time $FT_{\max}$, but we did not even start growing the green arc until time $FT_{\max}$.



In the initial state, the ring is full of garbage.

After one ring latency plus n Tframe times, all garbage values have been flushed out.

The Leader remains stuck, for at most one ring latency plus n Tframe times, until its own rotation counter has filled the ring.

Figure 6. Stabilization of the modified FDDI ring.

Let t_2 be the time at which the entire ring becomes green.

$$t_2 \leq FT_{\max} + L + n \cdot TT$$

Let r be the Leader's internal rotation counter at time t_2. The Leader's internal rotation counter has been r ever since time t_1, for some unique $t_1 \leq t_2$.

Imagine that all rotation counters initially in the ring at time 0 are baritone, but at time t_1 the Leader's rotation counter becomes soprano. As time passes, rotation counters that are "descended" from soprano rotation counters are also soprano. A soprano arc will steadily grow, starting at the Leader and proceeding around the ring. The soprano arc has the following properties:

1. All rotation counters within the soprano arc have the value r.
2. The soprano arc contains *all* green rotation counters with the value r.
3. The soprano arc contains at most one token.
4. Within the soprano arc, all phase bits upstream of the token are set, and all phase bits downstream of the token are unset. If the arc contains no token, then all phase bits are set.

At time t_1, when the soprano arc contains only the Leader, the properties must hold. In particular, property 2 holds because the Leader's rotation counter changes only by incrementing, and the counter size (12 bits) was deliberately made large enough that it cannot wrap around before time t_2.

We know that the Leader's rotation counter will not change between times t_1 and t_2, and after that, it cannot change until a Rotation Counter equal to r arrives, which cannot happen until the soprano arc engulfs the entire ring at time t_3. We are therefore assured that until time t_3, the Leader will not introduce any new rotation counter values into the soprano arc that would violate property 1. And because only the Leader is able to generate new rotation counter values, property 2 will remain true as well.

The Leader emits set Phase Bits when it lacks a token, and unset Phase Bits when it has a token, so it respects property 4. Non-Leaders without tokens repeat and copy all Rotation Counters and Phase Bits, so they cannot violate any of the properties.

The token-holder within the soprano arc emits only unset Phase Bits, respecting property 4. If it issues a Tframe, Phase Bits will be set upstream of the Tframe and unset downstream of the Tframe, until the token is captured.

The only chance left to violate any of the properties is at the frontier of the arc. When the message at the frontier encounters a station, the station copies the Rotation Counter and Phase Bit, thus joining the arc. There are four cases:

- The station had a token, and the message has a set Phase Bit. The station now holds the only token in the arc.
- The station did not have a token, and the message has a set Phase Bit. The station may have a token now, in which case it holds the only token in the arc.
- The station had a token, and the message has an unset Phase Bit. The station immediately issues a Tframe.
- The station did not have a token, and the message has an unset Phase Bit. The station still has no token.

In all four cases, properties 3 and 4 are preserved. Summing up, all four properties are preserved until the soprano arc engulfs the entire ring at time t_3. By the same argument used to time the growth of the green arc, we have:

$$t_3 - t_1 \leq L + n \cdot TT$$

Which implies that:

$$t_3 \leq FT_{\max} + 2(L + n \cdot TT) \doteq 5.75\,\mathrm{ms}$$

At time t_3, the properties apply to the entire ring. If there is no token in the ring, the message arriving at the Leader has a set Phase Bit, so the Leader is acquiring a token. Therefore, at time t_3 the ring is in a good state.

Notice that recovery from the most common fault, a corrupted Tframe that was carrying a token, is very fast—the very next message will deliver the token. The Leader makes sure that the ring is packed tight with messages, so the extra delay incurred will be at most $FT_{\max}$.

7 Costs

Our modifications to the FDDI MAC call for longer Tframes (by 5 Symbols). An extra cost of 200 ns is incurred every time a token is captured. In the worst case token rotation, in which every one of 1000 stations captures the token, the added overhead is 200 μs. With Target Token Rotation Times (TTRTs) typically on the order of 10 ms, that is a 2% reduction in usable bandwidth.

We also increase the per-Frame overhead from 56 to 62 Symbols. For very small payloads (32 bytes = 64 Symbols) throughput is reduced by 4.8%. For maximum payloads (4475 bytes = 8950 Symbols) throughput is reduced by only 0.067%.

We add 27 Symbol times (1.08 μs) to the ring latency as a result of the Leader's buffer, a delay equivalent to 212 m of fiber. The significance of this increased latency depends, of course, on the circumference of the ring.

Another cost is increased complexity in the stations. Each must now include a register to hold a rotation counter and phase bit, plus a 27-Symbol buffer for use when it is the Leader, plus a small amount of extra logic.

8 Benefits

The modified protocol recovers from a lost token (the most common fault, usually caused by line noise) faster than the original protocol. The original protocol waits for TVX to timeout (2.5 ms), then reinitializes the ring using the Claim Token process (one ring latency, up to 1.6 ms), taking up to 4.1 ms total. In the modified protocol, the next message (whether Tframe or Frame or fragment) arriving after the corrupted Tframe will deliver the token, because it carries the same Rotation Counter and Phase Bit that were in the Tframe before it got corrupted. When the corrupted Tframe was sent, there may have been a partial Frame just upstream of the sending station (a Frame that the sending station had previously sent, which had circulated around the ring and was then being stripped). The next message, which might be a Tframe created by the Leader, or a fragment resulting from another stripped Frame, can therefore lag behind the corrupted Tframe by as much as one maximal Frame time (0.36 ms). The average lag time will be half the average Frame time (0.02 ms for 512-byte payloads).

The original protocol can take *arbitrarily long* to recover from multiple tokens, which can be caused by memory corruption or line noise. The modified protocol can recover from any state[6], including one with multiple tokens, in two rings latencies (up to 3.23 ms) plus two Tframe times for each station (up to 2.16 ms), plus one maximal Frame time (0.36 ms), a total of at most 5.75 ms.

The Valid Transmission Timer (TVX) is not strictly needed even in the original protocol, but is included because it provides for faster recovery than the Late Counter in most cases (but TVX does not catch as many cases as the Late Counter, see Section 3.2). The modified protocol already recovers faster than TVX for the most common fault, so it is probably desirable to reduce complexity by eliminating TVX.

9 Simulation

We have written a simulator for the modified FDDI MAC that performs all actions down to the Symbol level, except for the following simplifications:

- There are no priorities—stations always use priority 0 for all frames. (This is a mechanism that affects only the timing policies for the capture and release of tokens, and does not concern us.)
- 16-bit addresses are not supported. (In practice, 48-bit addresses are preferred anyway.)
- Ignore_THT and MAC_Reset (conditions imposed from a higher layer) are not supported.
- Frame payloads, also known as Service Data Units (SDUs), are all 0s (we are not interested in the data being carried in the frames).
- The Frame checksum is sent as 0, and not checked upon receipt.
- The 8 redundant bits in the Rotation Counter are sent as 0, and not checked upon receipt. (This only makes the simulation more susceptable to simulated transmission errors.)

We have not conducted an extensive series of tests designed to analyze the performance of the protocol. We have, however, conducted a few tests to check whether the protocol performs roughly as we expect, mainly to provide some confidence that the simulator is correct (and hence, to provide additional assurance that the protocol is correct).

[6]Provided that there is exactly one Leader, see Section 5.1.

In our test runs, we used a ring of 10 identical stations. Each had a synchronous allocation of 5% of the TTRT, which was 8 ms. Each station had a source of synchronous traffic with an average rate of 2500 kbps, bundled into SDUs whose lengths were exponentially distributed with mean 64 bytes. Each station also had a source of asynchronous traffic with an average rate of 7500 kbps, bundled into SDUs whose lengths were exponentially distributed with mean 256 bytes. The interarrival times of the SDUs were also exponentially distributed (with mean determined by the data rate). Each station addressed its frames to the station exactly opposite it in the ring. Adjacent stations were linked by 1 km of fiber (with a delay of 5.085 ns per meter).

After a Leader had been elected, all the stations had almost all of their state information[7] randomized, and the links were filled with random symbols. In all four trials, the ring stabilized in about one ring latency. More precisely, each stabilization began with a small delay (between 0.24 μs and 1.48 μs) until the Leader issued the first Tframe, after which the Leader's unique rotation counter value raced around the ring, being copied by all stations along the way, until it arrived back at the Leader, at which point the global state was valid once again. The time for this circumnavigation was exactly one ring latency (51.9 μs, the delay through 10 km of fiber and the Leader's 27-Symbol buffer) in three of the trials, and in the fourth it was longer by just one Tframe transmission time (1.08μs), because the Rotation Counter value had to push its way through a token holder.

In another series of tests on the same ring configuration, bursty noise was introduced on one of the links to test the recovery time for lost tokens. The time between bursts was distributed exponentially with mean 1 μs, and the length of each burst was distributed exponentially with mean 100 ns (2.5 symbol times). We observed four tokens become lost due to corruption, causing the arrival of the token at the next station to be delayed until the next message. The delays ranged from 25 to 54 symbol times (1.0 μs to 2.2 μs). The delays would have been longer if the Frames had been larger; a maximal Frame of 9000 symbols could cause a delay of 360 μs.

10 Conclusions

We have demonstrated that stabilizing protocols can be useful for real networks. Our modified FDDI MAC token-passing protocol requires fairly minor changes to the existing specification. The advantages—increased robustness, like the ability to recover from multiple tokens, and faster recovery from the most common fault, a lost token—outweigh the costs of slightly reduced throughput and slightly increased complexity.

In the process of addressing the concerns of a real protocol, we have learned to solve problems that do not arise in the abstract model. For example, a real ring has both a ring latency (short) and a token rotation time (longer), two very distinct concepts, whereas the abstract model had just one notion of round-trip delay. This dichotomy led us to split the counter value into a separate rotation counter and phase bit, so that the stabilization time would be proportional to the short ring latency. The abstract model had stations retransmit their counter values, but in a real ring, this would require extra delay at every station, which led us to consolidate the burden of retransmission at just one station. In general, our goal has been to bridge the gap between elegant theoretical techniques for fault-tolerance and the needs of real networks.

[7] All counters, flags, and registers except those tied intimately to the non-stabilizing Claim Token process (TVX, T_Bid_Rc, T_Neg, T_Opr, T_Req, and Leader_Flag) and those merely concerned with token fairness (TRT, THT). Stations were not permitted to jump to states that begin the Claim Token process (CLAIM_TK and TX_BEACON).

References

[ANS87] American National Standards Institute. *Fiber Distributed Data Interface (FDDI) Token Ring Media Access Control (MAC)*, November 1987. ANSI X3.139-1987.

[Dij74] E. W. Dijkstra. Self stabilization in spite of distributed control. *Communications of the ACM*, 17:643–644, 1974.

[Jai94] Raj Jain. *FDDI Handbook: High-speed Networking Using Fiber and Other Media*. Addison-Wesley Publishing Company, Reading, MA, 1994.

[Per83] Radia Perlman. Fault tolerant broadcast of routing information. *Computer Networks*, December 1983.

[Ros81] E. C. Rosen. Vulnerabilities of network control protocols: An example. *Computer Communications Review*, July 1981.

[Var94] George Varghese. Self-stabilization by counter flushing. In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, August 1994. http://www.ccrc.wustl.edu/~varghese/PAPERS/cflushfinal.ps.Z.