The FDDI MAC Meets Self-Stabilization

Adam M. Costello
George Varghese

Supported by NSF Grant NCR-9405444
and the IPP Consortium

Motivation

Counter Flushing

- theoretical self-stabilization technique
- atomic send & receive
- token ring stabilizes in two “round-trip times”

FDDI MAC

- real-life token ring protocol
- *cut-through* forwarding
- *ring latency* vs. *token rotation time*

Implementing counter flushing in FDDI provides:

- Proof that the theoretical protocol can work in practice.
- Insight into the needs of real token rings, not accounted for in the theoretical protocol.
- Experience gauging the impact of self-stabilization on performance and complexity.

Outline

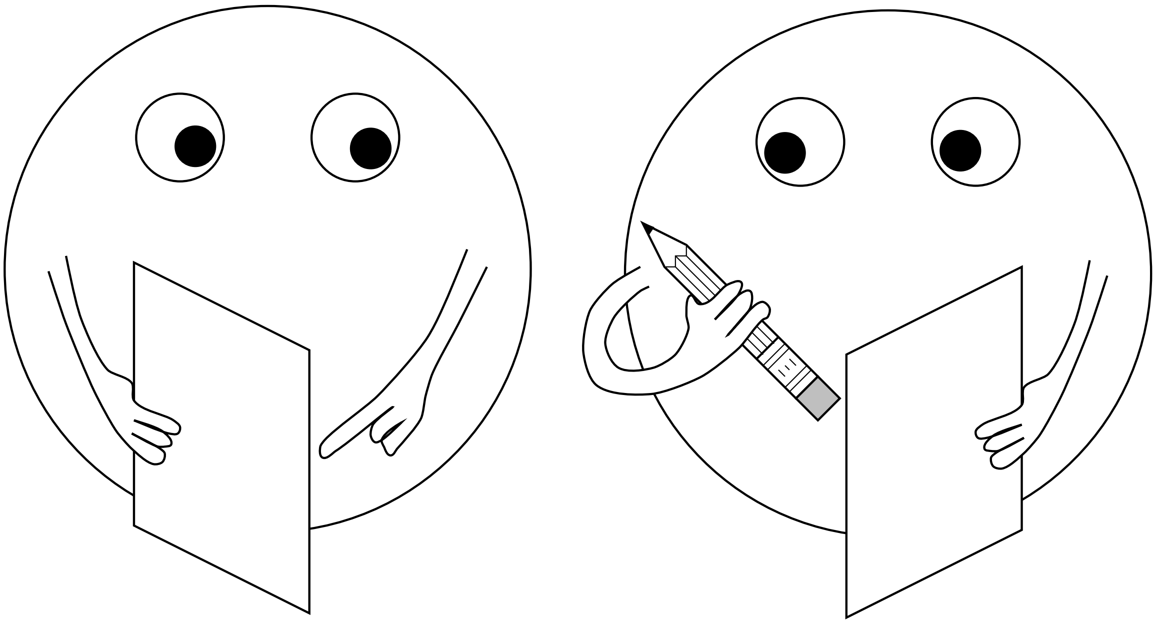
- counter flushing token ring
- FDDI MAC token ring
- refined counter flushing for real rings
- modified FDDI MAC (self-stabilizing)
- conclusions

Counter Flushing

One technique to achieve self-stabilization is *counter flushing*.

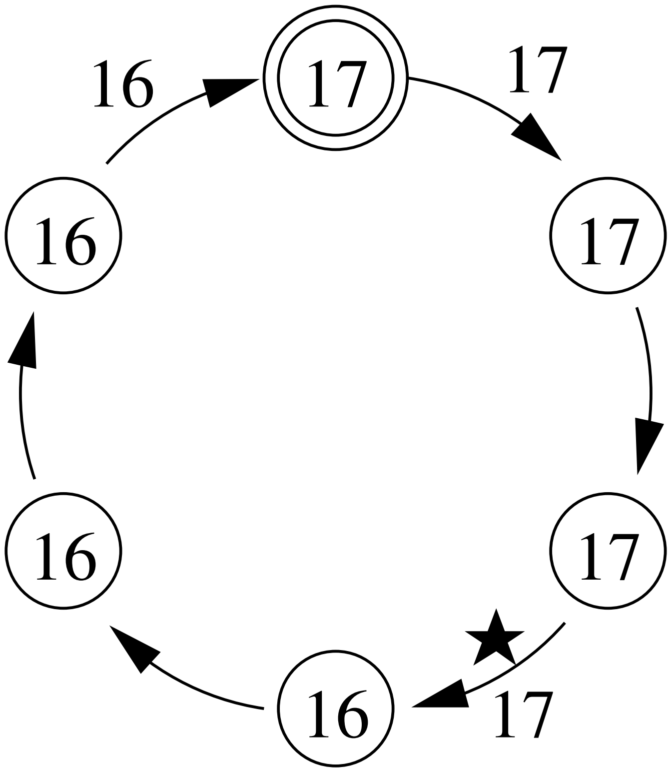
Nodes periodically inform neighbors of “where” in the execution they are.

Nodes who disagree change themselves to re-sync.



There is a “leader” who is always right. It never re-syncs, but it does do sanity checking on incoming messages.

Token Passing with Counter Flushing



"Where in the execution" is the number of times a node has held the token.

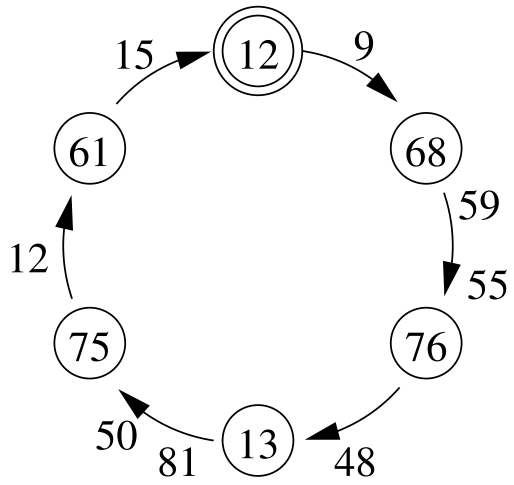
All nodes periodically send their counters.

Non-leaders always copy incoming counters.

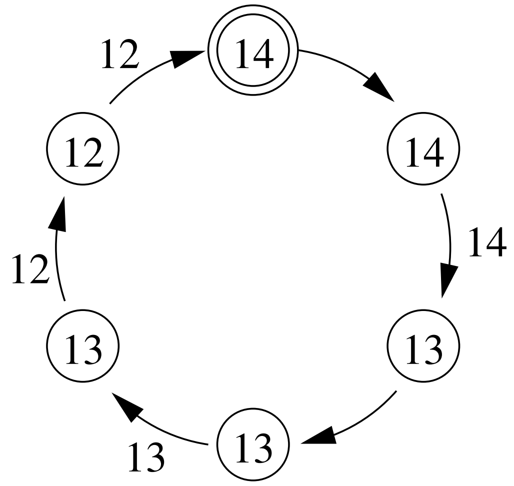
The leader increments when an incoming counter matches its own.

Links must be FIFO, and the modulus must be large enough.

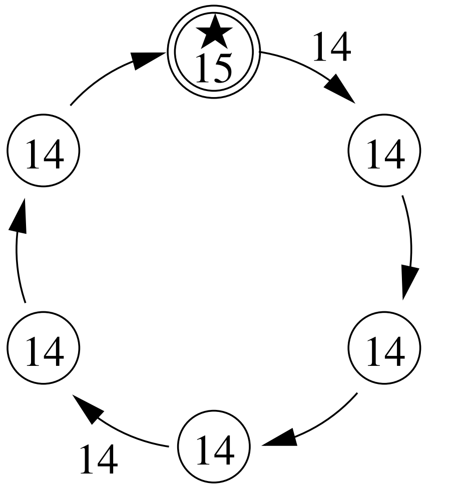
Stabilization of Token Ring



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



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



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

FDDI MAC

The token-holder:

- Sends Frames (data) for a bounded time.
- Then sends a Token message.

Non-token-holders:

- *Repeat* arriving Frames (*cut-through* forwarding).
- Frames from self are *stripped* (not repeated).
- Incoming Token repeated or *captured* (not repeated).

Recovery from lost token—a station resets the ring if:

- No message seen, TVX timeout ($\approx$ max ring latency).
- No token seen for two token rotation times.

Recovery from multiple tokens:
By chance only! (if tokens collide)

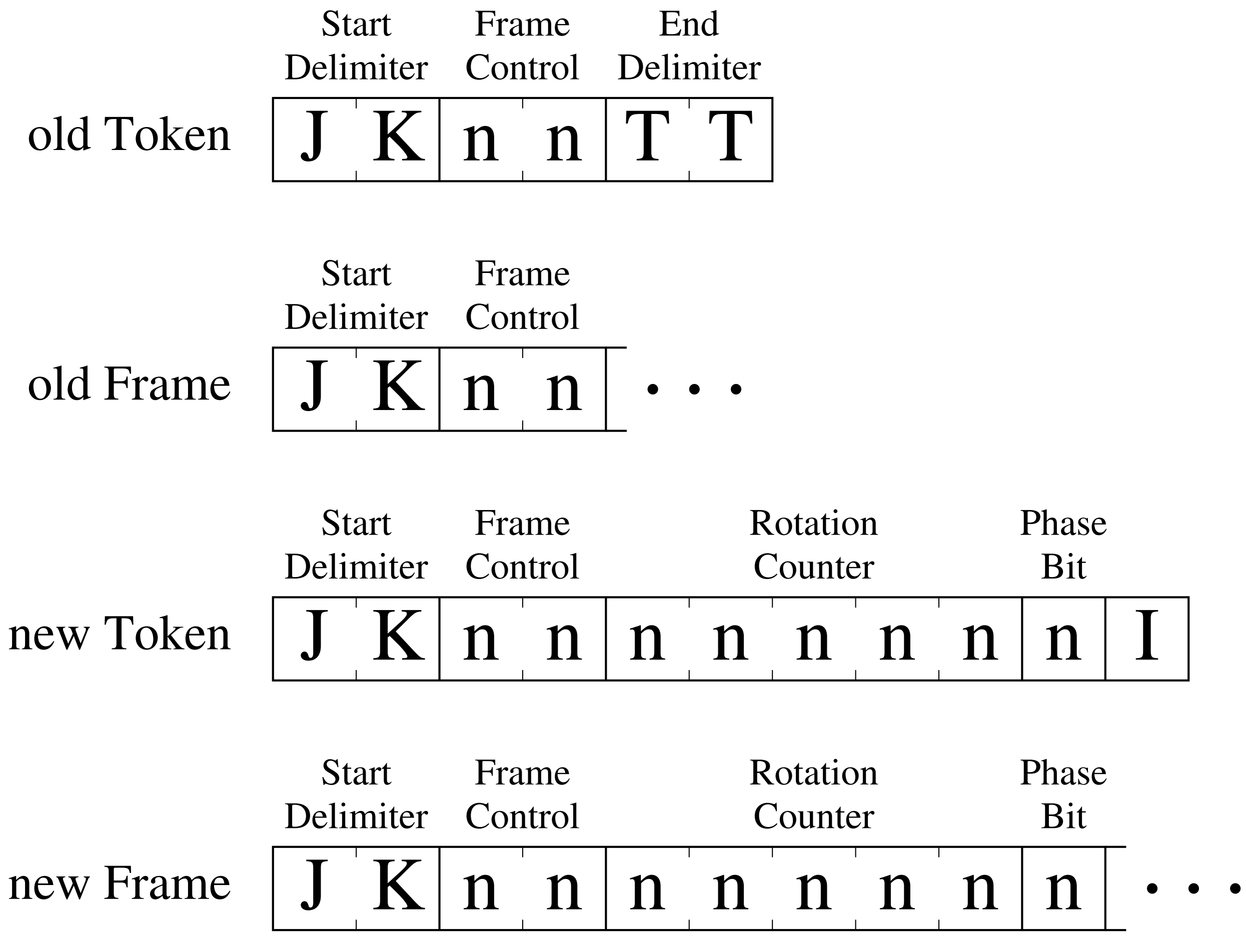
Theory vs. Practice

- **Problem:** Periodic retransmission requires a buffer between input and output. Unacceptable latency.
- **Solution:** Only the Leader retransmits; other nodes let counter values pass through.

- **Problem:** Token rotation time can be *much* greater than ring latency. We want counter value to pass through even when token does not.
- **Solution:** Add a *phase bit* which can be cleared to prevent a message from conveying the token.

- **Problem:** How to choose the Leader?
- **Solution:** *(not implemented)* Self-stabilizing leader-election algorithms exist. For now, we use the Claim Token process.

Message Formats



Tokens (Tframes) look like, and are treated like, Frame fragments.

Any message with set Phase Bit *may* convey the token.

Station Behavior

Each node contains an internal rotation counter and phase bit.

To relinquish the token quickly:
Copy internal rotation counter and phase bit into a new Tframe.

To send data while holding the token:
Copy internal rotation counter into a new Frame, but leave the Phase Bit clear.

Non-Leader Behavior

Always copy the incoming Rotation Counter and Phase Bit into the internal register (if error-free).

If register changes and phase bit is now set, optionally capture token by clearing the Phase Bit in the outgoing (repeated) message.

When there is nothing to send or repeat, emit Idle Symbols (I).

Leader Behavior

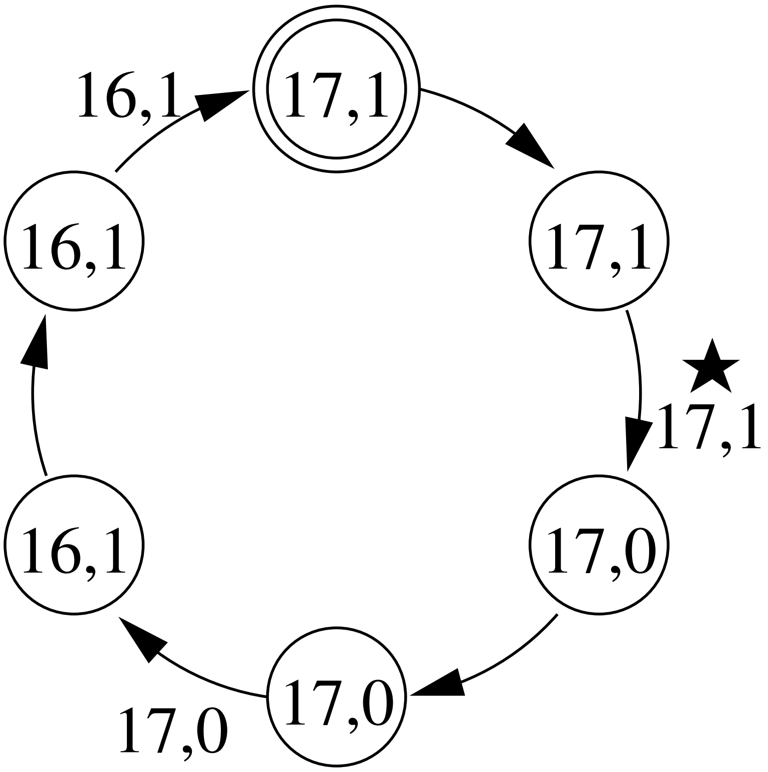
Always copy the internal rotation counter and phase bit into repeated messages.

If that caused no change, we have the token, increment internal rotation counter.

Leader's internal phase bit is *always* 1.

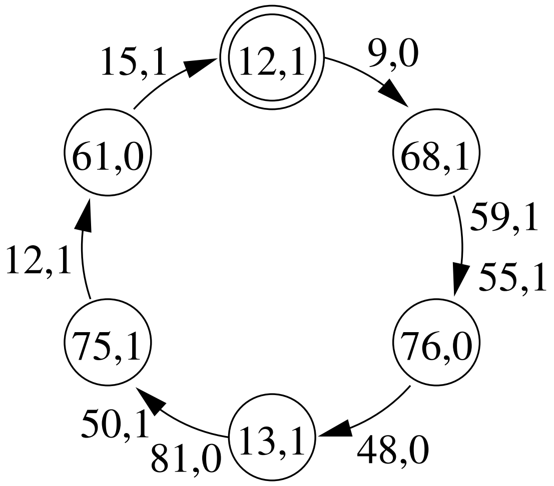
When there is nothing else to send or repeat, send Tframes. (Requires buffer big enough to hold one Tframe.)

Ring Operation

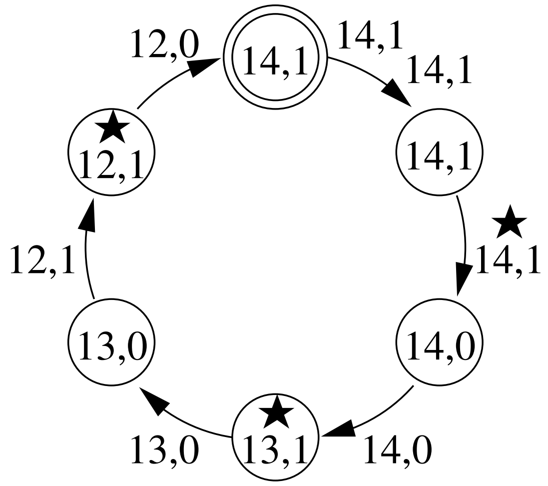


Three bands: New counter value propagates faster than the token.

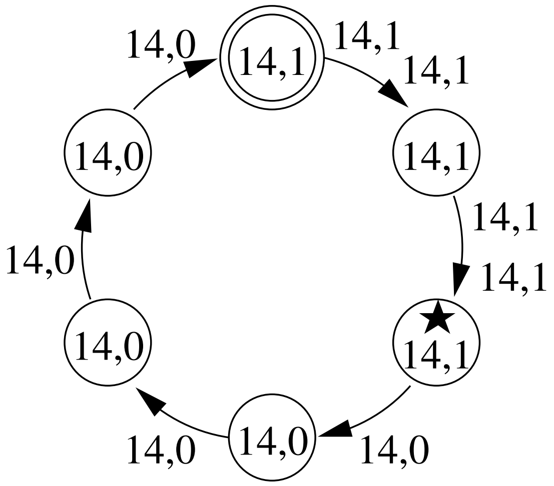
Ring Stabilization



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



After one ring latency plus at most N Token times, all garbage values have been flushed out.



The leader remains stuck until its own rotation counter fills the ring (at most one more ring latency plus N Token times).

Costs of FDDI Modification

- **Longer Token** (by 5 symbols) costs 200 ns per capture. For typical TTRT of 10 ms and maximal ring of 1000 stations, bandwidth reduced up to 2%, proportionately less for smaller rings.

- **Longer Frames** (by 6 symbols) costs 240 ns per Frame. For payload lengths of 512 bytes, bandwidth is reduced 0.6%.

- **Complexity** has increased somewhat, but not much compared to the parts of the FDDI MAC left untouched.

Benefits of FDDI Modification

- **Faster recovery** from lost tokens:

  old: TVX timeout plus ring latency
  (2.5 ms–4.1 ms, 2.6 ms typical)

  new: Frame tail time
  (0.36 ms max, 0.02 ms typical)

- **Robustness**—recovery from arbitrary state:

  old: maybe never

  new: 5.75 ms max, 0.1 ms typical

- **Simplicity:** TVX can optionally be eliminated.

FDDI Simulation

Simulator performs virtually all actions of the new FDDI MAC protocol, down to the symbol level.

Implementation led to a few tweaks of the spec, e.g. the need for an Idle Symbol (I) at the end of a Token.

Results as expected:

- Recovery from a lost token is very fast.
  (1.0–2.2 μs in tests,
  360 μs for maximal 4500-byte Frames)

- Recovery from an arbitrary state typically takes just over one ring latency.
  (52 μs for a 10 km fiber ring)

Conclusions

Self-stabilization can be useful in a real network:

- minor changes to existing protocol
- more robust
- faster recovery

Applying theory to a real network exposed new problems with new solutions:

- ring latency $<<$ token rotation time $\Longrightarrow$ phase bit
- cut-through forwarding $\Longrightarrow$ only leader retransmits