8. Frame formats

8.1 Frame header format

The frame header includes 48-bit *destinationMacAddress,* and 48-bit *sourceMacAddress* fields, as illustrated in Figure 8.1. The *srcStrip* bit is the broadcast/multicast group bit, with the effect that multicast *destinationAddress* traffic is sources stripped.

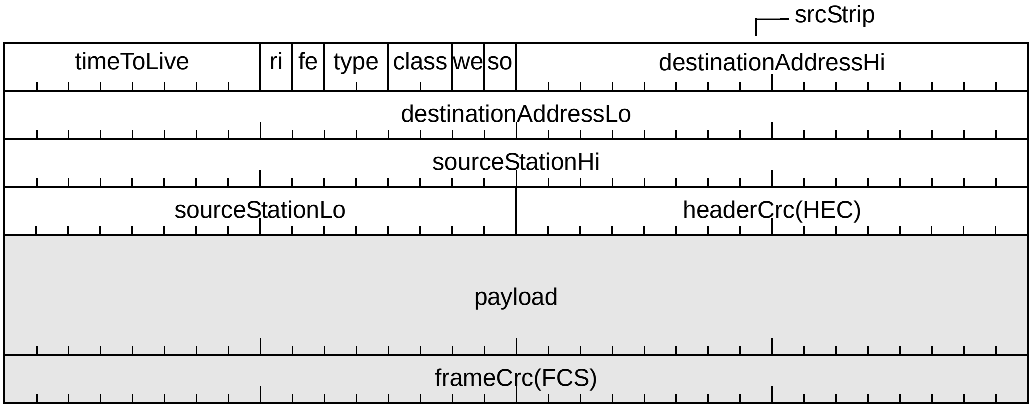


Figure 8.1—Frame header format

> ***Editors' Notes(DVJ):*** *To be removed prior to final publication.*
>
> *Relocating the sourceStrip bit to the sourceStationID field would support efficient remote unicast as well as multicast, although the use of this bit would be less elegant (token ring is rumored to use a similar coding).*

The 8-bit ***timeToLive*** field is set to an initial hop-count value and decremented when the frame passes through each station. The initial hop-count value is dependent on frame type, as listed in Table 8.3.

The ***ri*** (identity) bit values of 0 and 1 indicate the frame was sent on ringlet0 and ringlet1 respectively. The ***fe*** (fairness eligible) bit values of 0 and 1 disable and enable fairness, as specified in 8.1.2.

The 2-bit ***type*** field specifies the frame format and forwarding features, as specified in 8.1.1. The 2-bit ***class*** field values specify the class of RPR traffic, as specified in 8.1.2.

On the original ringlet, the ***we*** (wrap enabled) bit values indicate whether the frame is to be discarded or wrapped at the first edge. On the returning ringlet, the ***we*** (wrap enabled) bit values indicate whether the frame is to be wrapped or discarded at the first edge. See xx for details.

The ***so*** (strict ordering) bit values of 0 and 1 correspond to relaxed and strict ordering respectively. Relaxed ordered frames are delivered regardless of possible misordering or duplications irregularities; strict ordered frames are discarded rather than risk these irregularities.

The concatenation of the 16-bit ***destinationAddressHi*** and 32-bit ***destinationAddressLo*** fields forms the 48-bit *destinationAddress*.

The concatenation of the 32-bit ***sourceStationHi*** and 16-bit ***sourceStationLo*** fields forms the 48-bit *sourceStationID*, the MAC address of the initial or intermediate RPR source.

Frame processing is based on an implied *destinationStationID* field, whose derivation is dependent on the *srcStrip*-bit value:

a) *srcStrip*=0. The *destinationStationID*=*destinationAddress* and the frame is unicast.
b) *srcStrip*=1. The *destinationStationID*=*sourceStationID* and the frame is flooded.

8.1.1 Control type format

The 2-bit ***type*** field specifies the frame format, as specified in Table 8.1**.**

Table 8.1—Frame *type* field values

Value	Name	Description
0	FT_CONTROL	Control frame
1	FT_LOCAL	Directed local frame
2	FT_DIRECT	Global 4-address unicast frame
3	FT_FLOOD	Global 4-address multicast frame

8.1.2 Control *class* format

The 2-bit ***class*** field and the ***fe*** (fairness eligible) bit are combined to specify the class of RPR traffic, as specified in Figure 8.2. The distinct FC_DISCOVER class identifies FC_FAIRNESS frames that start with *timeToLive*=255, namely protection and fairness frames.

Table 8.2—*class* field values

class	fe	Name	Description
0	0	—	Reserved
	1	FC_CLASS_C	ClassC opportunistic weighted fairness
1	0	FC_CLASS_B0	Class-B committed information rate (allocated)
	1	FC_CLASS_B1	Class-B excess information rate (like CLASS_C)
2	0	FC_CLASS_A1	SubclassA1 (STQ option generated) traffic
	1	—	Reserved
3	0	FC_CLASS_A0	SubclassA0 (baseline) traffic
	1	—	Reserved

8.1.3 Frame encodings

The encodings corresponding to each of the control and data frames are listed in Table 8.3 and described below. Within this table, *hops* represents the number of hops between source and destination stations..

Table 8.3—Initial control field values

type	Format	srcStrip	timeToLive	controlType	Row	Frame type
FT_CONTROL	control (see 8.3)	0	hops	4	1	Echo control
				5	2	Flush control
	compact (see 8.2)	1	1	0	3	Idle filler
			255	1	4	Fairness control
				2	5	Protection control
				3	6	Discovery control
FT_LOCAL	local (see 8.4)	0	hops	—	7	Unicast local data
		1	"	—	8	Multicast local data
FT_DIRECT	global (see 8.5)	0	hops	—	9	Directed global data
FT_FLOOD	"	"	"	—	10	Flooded global data

Row 8.3-1: The echo frame is triggered by the client, to validate the operation of data paths between the source and specified destination stations. See xx for details.
Row 8.3-2: The flush frame is triggered by the client, to confirm the completion of previously initiated frame transfers. See xx for details.

Row 8.3-3: The idle frame is transmitted during interframe gaps, to limit the receiver’s effective frame bandwidth to slightly less than the link bandwidth. See xx for details.
Row 8.3-4: The fairness messages are transmitted by congested stations, to temporarily inhibit transmissions from less congested stations. See xx for details.

Row 8.3-5: The protection frame is transmitted after link-status changes, to communicate this topology change to other stations. See xx for details.
Row 8.3-6: The discovery frame is transmitted periodically by each station, to inform each station of the noncritical characteristics of the others. See xx for details.

Row 8.3-7: The unicast frame sends data between RPR-local stations, with minimal 16-byte overhead.
The *destinationStationID* and *sourceStationID* identify local RPR stations. See xx for details.
Row 8.3-8: The multicast frame floods data to other RPR-local stations, with minimal 16-byte overhead.
The *destinationStationID* is multicast; the *sourceStationID* identifies a local RPR station. See xx for details.

Row 8.3-9: The directed frame sends data between networked stations, with extended 28-byte overhead.
Distinct *destinationStationID*, *sourceStationID*, *destinationMacAddress*, and *sourceMacAddress* fields are transported; *destinationStationID* and *sourceStationID* identify local RPR stations. See 8.4 for details.
Row 8.3-10: The flooded frame floods data between networked stations, with extended 28-byte overhead.
Distinct *destinationStationID*, *sourceStationID*, *destinationMacAddress*, and *sourceMacAddress* fields are transported; *destinationStationID* and *sourceStationID* identify local RPR stations. See 8.5 for details.

8.2 Compact frame format

The special frame format is implied by (*type*=FT_CONTROL, *srcStrip=1*); source stripping is assumed and the otherwise unused *destinationAddress* field carries dependent *parameters* information, as illustrated in Figure 8.2. The ***sourceStrip*** bit, whose value shall be 1, is specified in 8.1.

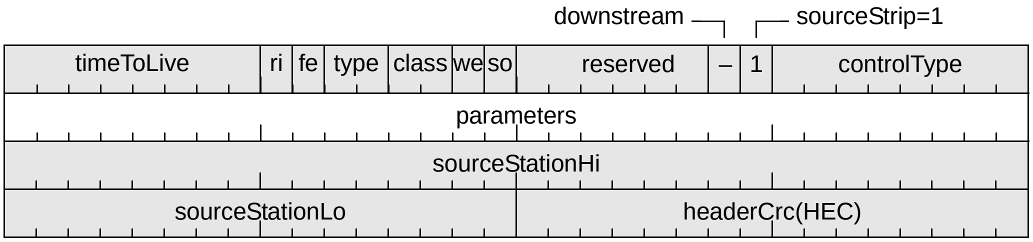


Figure 8.2—Compact frame format

The 7-bit ***reserved*** field shall be reserved. The ***downstream*** bit values of 0 and 1 correspond to source-stripped and neighbor-stripped addressing formats respectively.

The 7-bit ***controlType*** field distinguishes between distinct control frame types, as specified in Table 8.3. The 32-bit ***parameters*** field is controlType-field dependent and specified on a individual frame basis. The remaining fields are specified in 8.1.

NOTE—This use of a distinct downstream indication eliminates the need to allocate a special *destinationMacAddress* for this purpose, while providing space for an additional/adjacent 32-bit parameters field within the frame header.

8.3 Control frame formats

The control frame (*type*=FT_CONTROL, *srcStrip=0*) has the standard header and the payload starts with an identifier specified by this standard, as illustrated in Figure 8.3. The ***sourceStrip*** bit, whose value shall be 0, is specified in 8.1.

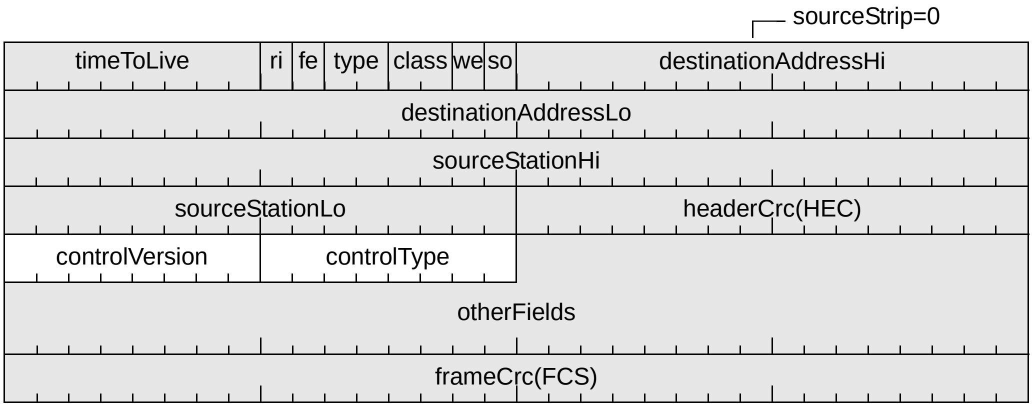


Figure 8.3—Control frame format

The 8-bit ***controlVersion*** field shall be zero (nonzero values shall be reserved for future revisions of this standard). The 8-bit ***controlType*** field specifies the form of control frame, as defined in Table 8.3.

8.4 Local frame format

The local frame (*type*=FT_LOCAL) has the standard header and the payload starts with an ethernet-standard identifier, as illustrated in Figure 8.4. The 8-bit *timeToLive* field through the 16-bit *headerCrc* field are specified in 8.1; the 48-bit *destinationAddress* field is further described below..

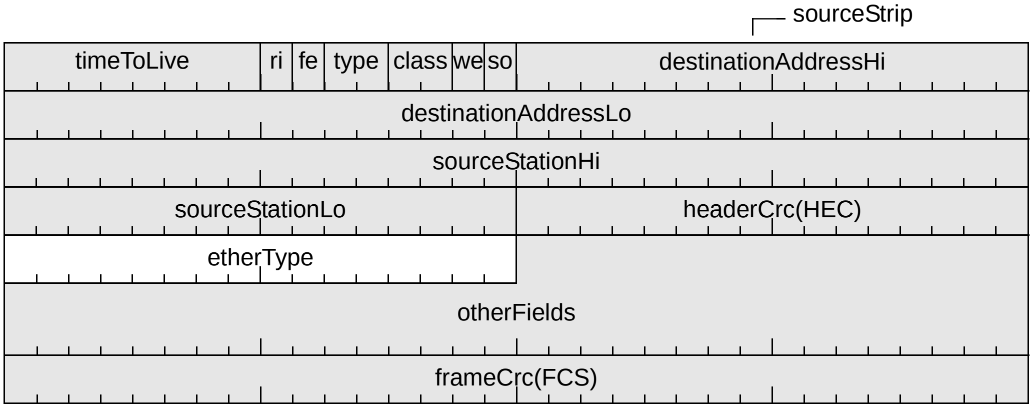


Figure 8.4—Local frame format

The meaning of the 48-bit *destinationAddress* field is dependent on the *srcStrip*-bit value:

a) *srcStrip*=0. The *destinationAddress* corresponds to a local RPR station and the frame is unicast.
b) *srcStrip*=1. The *destinationAddress* corresponds to a multicast address and the frame is flooded.

NOTE—This *srcStrip* encoding convention allows an RPR station to simply flood multicast frames to itself, with mimimal header-size overhead. Remotely-sourced flooded frames are required to use a less compact global-frame format (see 8.5).

The 32-bit ***sourceStationHi*** and 16-bit ***sourceStationLo*** fields are concatenated to form a 48-bit *sourceStationID* field that specifies the endpoint MAC source address.

The 16-bit ***etherType*** field shall indicate the type of data frame, selecting from values designated by the IEEE RAC.

8.5 Global frame formats

The extended frames (*type*=FT_DIRECT and *type*=FT_FLOOD) have the standard header, as illustrated in Figure 8.5. The 8-bit *timeToLive* field through the 16-bit *headerCrc* field are specified in 8.1; the 48-bit *destinationAddress* field identifies a local RPR station.

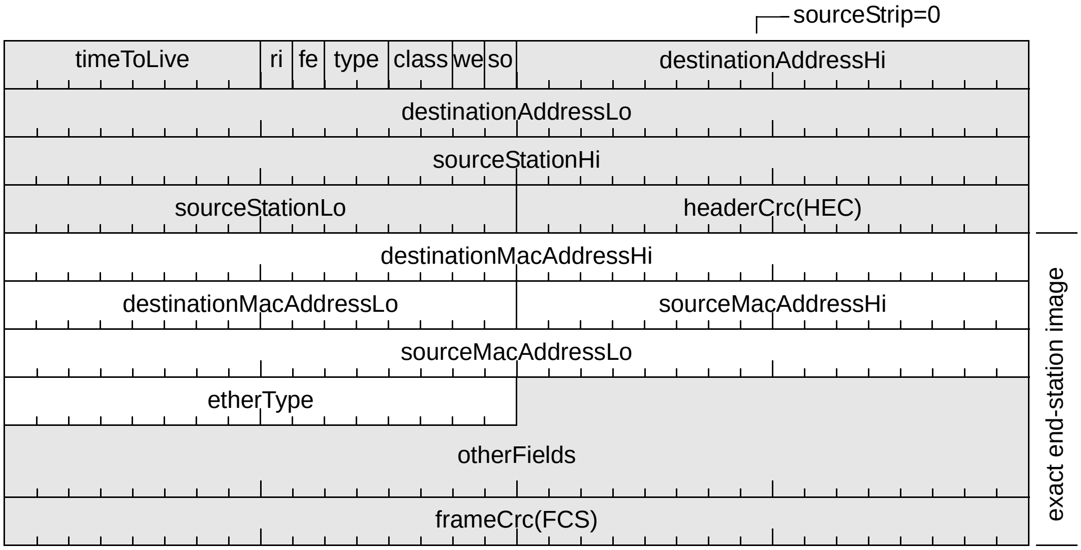


Figure 8.5—Global frame format

The 32-bit ***destinationMacAddressHi*** and 16-bit ***destinationMacAddressLo*** fields are concatenated to form the 48-bit *destinationMacAddress* field that specifies the endpoint MAC destination address.

The 16-bit ***sourceMacAddressHi*** and 32-bit ***sourceMacAddressLo*** fields are concatenated to form the 48-bit *sourceMacAddress* field that specifies the endpoint MAC source address.

The 16-bit ***etherType*** field shall indicate the type of data frame, selecting from values designated by the IEEE RAC.

NOTE—The payload values are identical to the image that is transferred to the endpoint ethernet station. The CRC isn't necessarily recalculated when passing through intermediate bridges, which would otherwise place the frame in a transient unprotected state. Thus, end-to-end error coverage is easily maintained.

Annex F.

(normative)

802.1D and 802.1Q Bridging conformance

Editors' Notes (DVJ): *To be removed prior to final publication.*

The following should be updated in the terms and definitions:

unidirectional flooding -- a frame forwarding transfer involving sending a flooding frame in the downstream direction, where that frame is directed to its sending station. The source-strip operation occurs at the source's upstream station, to avoid the unnecessary final-link transmission.

bidirectional flooding -- a frame forwarding transfer involving sending two flooding frames, one on each ringlet, where the each frame is directed to distinct adjacent stations.

remote unicast -- A transfer directed to a specific RPR station with the intent of forwarding a stripped version of that frame to a distinct remote node.

local unicast -- A transfer sourced-from and directed-to local RPR end stations.

flooded -- A multicast, broadcast, or unlearned remote-unicast frame sent from one station to all others.

run -- Each of the two datapaths that (when unwrapped) correspond to ringlet0 or ringlet1 respectively.

Editors' Notes (DVJ): *To be removed prior to final publication.*

The following is a list of recent changes:

The following changes (and several clarifications) were made after 2002Nov07 meeting review:
1) **Concern:** Some of the code has reminants of the frame.ws bit, which no longer exists.
   Action: Updated appropriately.
2) **Concern:** Discard conditions are not well defined.
   Action: All discard conditions are now listed, including the timeToLive==0.
3) **Concern:** Clearly identify the scope of influence due to the strict bit.
   Action: It has been noted that the destinationStation/timeToLive checks are:
   a) Only needed if passthru stations are supported.
   b) Only utilized for strict traffic (relaxed traffic can ignore these checks).

The following changes (and several clarifications) were made after 2002Nov08 meeting review:
1) **Concern:** The encoding of various frame types is not easily deduced from this writeup.
   Action: Another table has been added to clause #8, with row-by-row descriptions of each frame type.
2) **Concern:** The use of a class field to represent a frame type seems a bit funky.
   Action: A class code was previously used to specify frame type; this has been eliminated.
3) **Concern:** Optimization for unidirectional0 is inappropriate, since unidirectional1 is more efficient.
   Action: There is now only unidirectional flooding, stripped by the upstream-from-source station.
4) **Concern:** The bimodal complexity is increased by the need to identify adjacent midpoints.
   Action: The same endpoint is used in both transmissions, but a midpoint-neighbor station strips early.
5) **Concern:** Its hard to keep track of draft numbers, while things are changing so rapidly.
   Action: The print data is now automatically included in each page header.
6) **Concern:** The loss, copy, toss, and aged phases of frame processing are not well specified.
   Action: These phases are now well specified.

Editors' Notes (DVJ): *Change history: to be removed prior to final publication.*

The following changes were in response to comments received before the 2002Oct24 meeting:

1) **Concern:** Elimination of the CLASS_B0&CLASS_C distinction was undesirable.
   Action: The existing definitions of fe and class have been restored.
2) **Concern:** The distinction between remote-unicast and remote-flood should be maintained.
   Action: Types now include FT_CONTROL, FT_LOCAL, FT_DIRECT (extended unicast), and FT_FLOOD (extended flood).
3) **Concern:** The two types of flood removal, COPY and TOSS, are unnecessarily complex.
   Action: The distinct flood types has been eliminated. For unidirectional, the distinction between COPY and TOSS was already implied by the destinationStationID. For bidirectional, the less-efficient option (that needed this distinction) has been eliminated.
4) **Concern:** Having *timeToLive* start at 255 doesn't limit a frame's distance or support fairness.
   Action: The timeToLive value now specifies the distance-to-destionation, which reduces the frame-discard delays. To harmonize this change, a destinationStationID (rather than sourceStationID) check is used to detect for pass-through deletions.

The following changes were in response to comments received at the 2002Oct24 meeting:

1) **Concern:** A distinct command code should be provided to identify fairness frames.
   Action: One of the unused class fields was allocated for this purpose.
2) **Concern:** A comparison chart between this and the alternative proposal should be presented.
   Action: This comparison chart follows.
   The following changes were in response to comments received before 2002Nov04 emails:
3) **Concern:** A strict bit is useful for avoiding unnecessary precautionary losses.
   Action: Strict and relaxed verisons of steering are now supported.
   Since wrapping has minimal losses, and code space was tight, no equivalent wrapping bit is provided.
4) **Concern:** There is no need to penalize local RPR multicasts with an extended header overhead.
   Action: This has been accomplished while maintaining the integrity of the destinationStationID check. If the initial destinationStationID field is broadcast, the effective destinationStationID (for stripping purposes) is assumed to equal the sourceStationID field within the frame.

The following changes (and several clarifications) were made during 2002Nov06 editing updates, based on email discussions of concerns:

1) **Concern:** A strict bit is useful for avoiding unnecessary precautionary losses.
   Action: Strict versions of wrapping and steering are both supported.
2) **Concern:** Special frame formats should be well defined.
   Action: The idle, fairness, and discovery frames are defined and extensible.
3) **Concern:** The dSpan compensation at the return-run destination should be better described.
   Action: A better description has been provided, including a definition of pre-compensation for the following timeToLive adjustment (perhaps this should be phrased otherwise, for more clarity?)
4) **Concern:** The dSpan adjustment temporarily discard frames until the distance-to-edge is known.
   Action: To minimize transient sensitivities, the timeToLive is either:
   a) Compensated by the ringlet diameter, before the edge location is known.
   b) Compensated by the downstream edge distance, when the edge location is known.
   This reduces the frame discards (incorrect compensation doesn't actually creates errors).

F.1 Proposal comparison

At the 2002Oct03 meeting, the working group expressed its displaasure with proposal-P and proposal-Q proposals. This new proposal is therefore dubbed the after-death (AD) proposal, representing extensive revisions to addressed the concerns of the RPR working group. Another proposal represents mindful modifications (MM) of the working group discussions. Both proposal labels coincidentally contain the first-name initials of the two primary advocates. The term WH (white horse) refers to a late entrant. A comparison of proposal properties is contained in Table F.1

.

Table F.1—Proposal comparison

Property	Row	Proposal AD	Proposal MM	Proposal WH
Local unicast frame size	1	⇧16-byte	⇩18-byte	⇧16-byte
Local multicast frame size	2	⇧16-byte	⇩18-byte	⇧16-byte
Local relaxed unidirectional&bidirectional	3	⇧16-byte	⇩18-byte	⇧16-byte
Local strict unidirectional&bidirectional	4	⇩28-byte	⇧18-byte	⇩30-byte
Remote-source frame size	5	⇧28-byte	⇩30-byte	⇩30-byte
Reserved bits	6	~0.25 bits	3 bits	~0.25 bits
Idle frames	7	⇧standard compact frame	??	??
Discovery frames	8	⇧standard compact frame	??	??
Fairness frame	9	⇧standard compact frame	⇩special form&HEC	??
Wrap-enabled bidirectional flooding	10	⇧standard	⇩not robust	??
Steered correctness with bypass ????	11	yes	no	??
Checks timeToLive/stationID consistency	12	⇧destinationID	⇩sourceID	??
Four address equivalents provided (StationID & MacAddress pairs)	13	⇧yes	⇩no	⇩no
Extensible controlType in the header	14	⇧yes	many bits	⇩no
Distinct timeToLive discards and intended destination deletions	15	⇧yes	⇩no	⇩no
The timeToLive has hops-to-destination information for fairness and lifetime limits	16	⇧yes		??
Discovery purging is sufficient	17	⇧yes		⇩no
Additional passthru-check requirements	18	⇧Indexed consistency check		⇩CAM check
Independent scoping decisions	19	⇧Independent on each flow		⇩Committed
Per-frame strictness bit for reduced losses	20	☺ yes		
End-to-end coverage on remote frames	21	☺ yes		
Simple unidirectional flooding	22	☺ yes		

Row F.1-1: The AD unicast header is 16 bytes; the MM header is a slightly larger 18 bytes.
Row F.1-2: The AD multicast header is 16 bytes; the MM header is a slightly larger 18 bytes.
Row F.1-5: The size of The effective AD and MM global-multicast headers increases to 28 and 30 bytes respectively, due to additional *destinationMacAddress* and *sourceMacAddress* parameters within payloads.

Row F.1-9: The AD proposal utilizes a near-standard fairness frame format, where the *destinationStationID* transports fairness information; the MM proposal mandates a distinct fairness frame format to meet the perceived 16-byte fairness frame-size objective.
Row F.1-11: The AD proposal supports bidirectional flooding; the MM proposal does not (see F.6.2&F.6.3).
Row F.1-12: The AD proposal detects and logs errors when the *timeToLive*-specified location differs from the *destinationStationID*-specified location; the MM proposal does not.

Row F.1-13: All four network addresses (destinationStationID, sourceStationID, destinationMacAddress, and sourceMacAddress) are available in AD proposal; the MM proposal does not always provide them.

Row F.1-16: Both proposals set *timeToLive* to the hops between the source and destination stations, thus ensuring early and more reliable frame removals.
Row F.1-18: Both proposals have an additional indexed-table consistency-check requirement when passthrough stations are to be supported. However, no CAM lookups are required.
Row F.1-22: All proposals support simple unidirectional flooding (flooding to one’s self).

F.2 Flooding techniques

F.2.1 Multicast/broadcast forwarding

Editors' Notes (DVJ): *To be removed prior to final publication.*

The following discussion on multicast/broadcast forwarding probably belongs in the overview (with details in clause 6), but is being retained in this annex until new placement location is confirmed:

The most basic multicast/broadcast distribution techniques involves circulating a frame through all stations on the ring. The forwarding techniques for multicast/broadcast transfers are the same as those described for flooded frames, illustrated in F.3.1 through F.3.2.

NOTE—Stations are not expected to optimize the efficiency of multicast forwarding. To reduce complexities, they are expected either support unidirectional multicasts or to forward multicasts and flooded frames in the same fashion.

F.2.2 Flooding bridge transfers

Editors' Notes (DVJ): *To be removed prior to final publication.*

The following flooding text probably belongs in the overview, since its mostly informative.

Transparent bridging requires a form of one-to-many distribution called flooding. Flooding protocols (flooding, multicast, and broadcast) require the inclusion of additional information, beyond that included within the client-visible Ethernet frame. That information includes local *destinationStationID* and *sourceStationID* information. These identifiers assist in scoping the range of the flooding distribution and suppressing undesirable duplicates that might otherwise be generated.

Bridges use the *destinationStationID* and *sourceStationID* information to flood (a flood is a type of broadcast) remote frames for delivery to all all bridge clients, as illustrated in the left side of Figure F.5. Flooding involves transmissions between a single source station and all other stations.

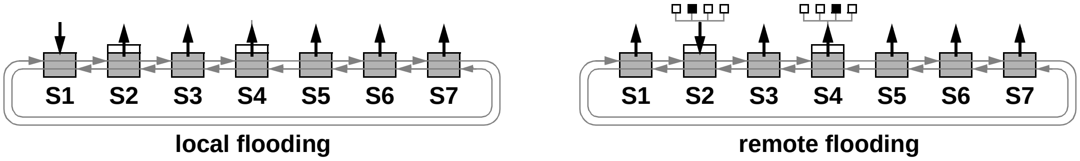


Figure F.5—Basic bridge flooding

With flooding, a frame is placed on the ring by the source, copied by intermediate stations, and stripped at the destination (depending on the method, the frame may also be copied in the destination). Flooded frames may pass through an additional multicast filter, which selectively rejects uninteresting multicasts, but this action is decoupled and independent from the flood protocols of the RPR ringlet.

Basic-bridging stations maintain simplicity by always flooding, as illustrated in Figure F.5. Although no spatial reuse is possible, this avoids overheads associated with maintaining and utilizing RPR forwarding tables.

F.2.3 Unicast considerations

Editors' Notes (DVJ): *To be removed prior to final publication.*

The following informative belongs in the overview. While enhanced bridging is not part of this specification, background information is useful to ensure that code space is sufficient for enhanced bridging extensions.

Local stations improve efficiencies by directing local-unicast traffic to the affected station, rather than flooding this traffic to all others, as illustrated in the left side of Figure F.6. The determination of whether to use flooded or unicast frames is based on a comparison of the frame's *destinationMacAddress* with the RPR topology database: a unicast is used if a local *stationID* matches the same *destinationMacAddress*; a flood is used otherwise.

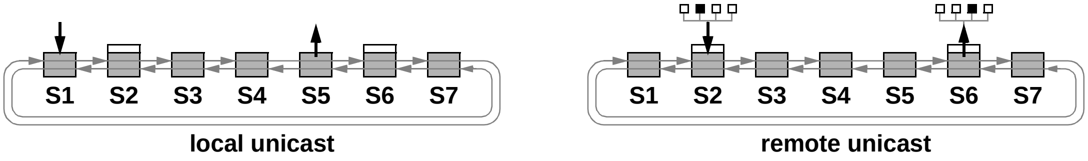


Figure F.6—Enhanced bridge unicasts

Enhanced-bridging stations are expected to improve efficiencies by maintaining and utilizing RPR forwarding tables, so that remote frames can also be unicast, as illustrated in the right side of Figure F.6. The learn improves link utilization, due to the frame's shortest-path unicast nature.

Ordering constraints mandate that flooded and related remote-unicast transfers flow over the same path. The term *related* refers to frames with an identical set of *{sourceMacAddress/destinationMacAddress,VLAN}* identifiers. Flowing over the same path is necessary to maintain ordering, without invoking an inefficient flush between floods and related remote-unicast transfers.

For unidirectional flooding, the potential performance impact of this ordering constraint can be severe, in that the worst case path-length nearly doubles over that associated with bidirectional flooding. To avoid that potential performance impact, enhanced bridges are expected to support bidirectional flooding.

F.2.4 Flooding flow notation

A variety of remote-transfer flows are described in following subclauses, as illustrated in Figure F.7. Downward and upward arrows identify client-to-MAC and MAC-to-client transfers respectively. Downward end-of-flow curves identify locations where frames are stripped. The *x* marker at the end of an error identifies locations where frames are discarded, due to detected inconsistency errors.

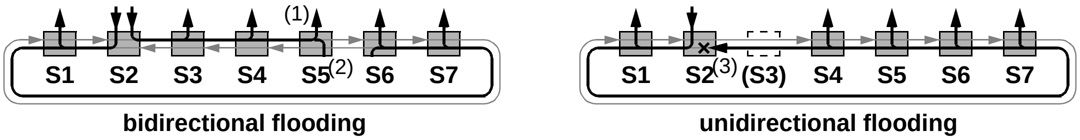


Figure F.7—Flooded operations

F.3 Flooding alternatives

Several flooding alternatives are provided, as illustrated in Table F.2. Flooding entries within this table are ordered by complexity: the first-through-last entries are the most-through-least complex and most-through-least efficient (from a link-bandwidth utilization perspective). Stations may use any of the four listed alternatives. Within this table, the *Reference* column provides a cross-reference to the applicable descriptive subclause.

Table F.2—Flooding alternatives

Name	ringlet	DSID	Reference	Description
unidirectional	0 or 1	sourceStationID	F.3.1	One flood, returned to the source
bidirectional	0 and 1	midPoint	F.3.2	Two floods, to a midpoint stations

The unidirectional flooding is the simplest, since only hops-to-self topology database information is needed. To avoid saturating one ringlet, simple load-balancing techniques could involve hashing the *sourceMacAddress*, *VLAN*, and *priority* fields to consistently select between westside and eastside paths.

Bidirectional flooding efficiently balances the load on both ringlets, but requires maintenance and use of a known midpoint location.

F.3.1 Unidirectional flooding

F.3.1.1 Unidirectional loop flooding

Unidirectional flooding involves either a westside or eastside transmission directed to the source station, as in the left and right side of Figure F.8 respectively. The frame is addressed to the source, but nominally stripped at the upstream neighbor, to avoid the unnecessary final-link transmission: with the effect of nullifying the S2-to-S3 and S1-to-S2 transmissions in the left and right side sides of Figure F.8 respectively.

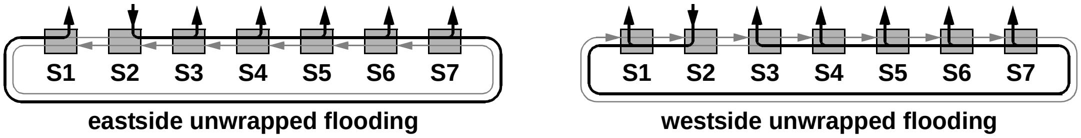


Figure F.8—Unidirectional loop flooding

F.3.1.2 Unidirectional wrap-protection flooding

Unidirectional flooding with wrapped protection involves either a westside or eastside transmission directed to the source station, as in the left and right side of Figure F.9 respectively. The wrapped flooding operation relies on the wrap capability at the endpoints.

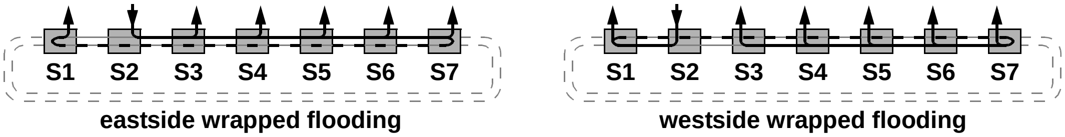


Figure F.9—Unidirectional wrap-protection flooding

The decision to toss (rather than copy) the stripped flooded frame is based on the stripped-at-source property of the frame, as validated by equal *frame.destinationStationID*, *frame.sourceStationID*, and *mac.stationID* values.

F.3.1.3 Unidirectional steer-protection flooding

Unidirectional steer-protection flooding involves concurrent east and west transmissions of the same source-stripped frame. These steered transmissions are stripped at the edge points before reaching their specified source-stripped location, as illustrated in Figure F.10.

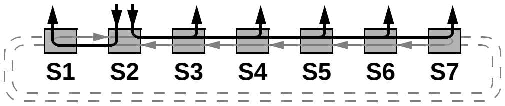


Figure F.10—Unidirectional steer-protection flooding

F.3.2 Bidirectional flooding

F.3.2.1 Bidirectional loop flooding

Bidirectional wrap-protection flooding involves concurrent transmissions on both ringlets, directed to the same target station, as illustrated in Figure F.11. The frame is addressed to a common target, but the ringlet1 frame is nominally stripped at S6, one station upstream of the target S5 station.

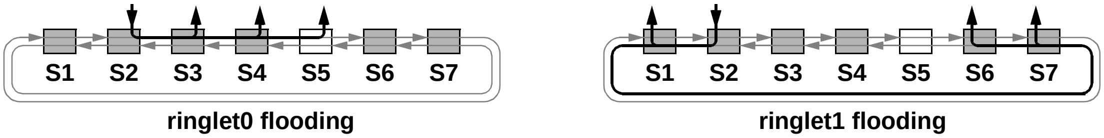


Figure F.11—Bidirectional loop flooding

F.3.2.2 Bidirectional wrap-protection flooding

Bidirectional wrap-protection flooding involves concurrent transmissions on both ringlets, again directed to the same target station, as illustrated in Figure F.12. The frame is addressed to the common S5 target, but the ringlet1 frame is nominally stripped at S6, one station upstream of the target station..

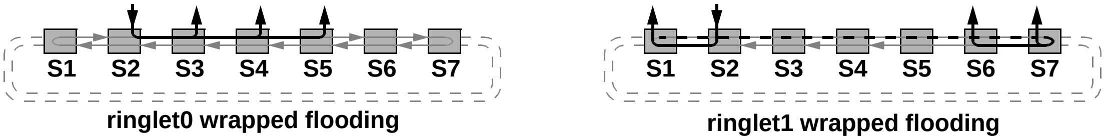


Figure F.12—Bidirectional wrap-protection flooding

F.3.2.3 Bidirectional steer-protection flooding

Unidirectional steer-protection flooding involves concurrent east and west transmissions of the same source-stripped frame, as described in F.3.1.3.

F.3.3 Flooding constraints

Based on its flooding constraints, clause 6 of IEEE Std 802.1D-1998 architecture is summarized as follows:

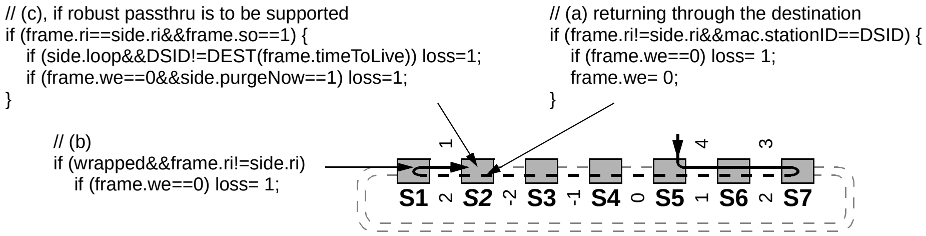
a) Loss: A frame may be lost due to many reasons including topology change (subclause 6.3.2).
b) Misordering. Frame misordering is not allowed under any conditions (subclause 6.3.3).
c) Duplication. Frame duplication is not allowed under any conditions (subclause 6.3.4).

The misordering constraint can be met by having the client force a flush of in-progress transimissions before topology changes are allowed to affect the forwarding parameters (see F.7.2). In normal operation, the duplication constraint is met by flooding in nonoverlapping directions, as described in F.3.1 and F.3.2. In abnormal operation, such as during topology changes the duplication constraint mandate consistency checks based on the *destinationStationID* and *timeToLive* values, as described in F.4.1.

F.4 Frame recieve processing

F.4.1 Early consistency checks

Early consistency checks are performed before the frame-copy decison is committed, as illustrated in Figure F.13.



```
// (c), if robust passthru is to be supported
if (frame.ri==side.ri&&frame.so==1) {
    if (side.loop&&DSID!=DEST(frame.timeToLive)) loss=1;
    if (frame.we==0&&side.purgeNow==1) loss=1;
}
```

```
// (a) returning through the destination
if (frame.ri!=side.ri&&mac.stationID==DSID) {
    if (frame.we==0) loss= 1;
    frame.we= 0;
}
```

```
// (b)
if (wrapped&&frame.ri!=side.ri)
    if (frame.we==0) loss= 1;
```

Figure F.13—Early consistency checks

For wrapped rings, the consistency checks involve (a) confirming the destination’s presence on a wrapped return run. When (b) wrapping back to the originating run, unconfirmed (*frame.we==1*) frames are discarded; in the absence of terminal destination station, there is no point in sustaining these frames.

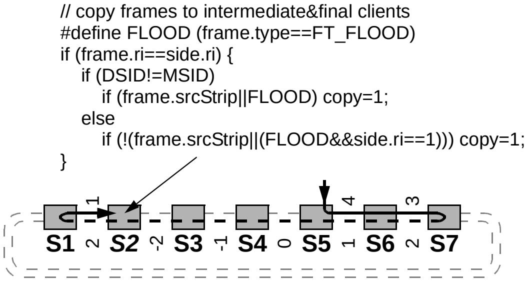
Other consistency checks (c) are performed when the frame passes through stations, to delete possibly misordered or duplicated frames. Two types of checks are performed, as follows:

a) The frame’s implied DSID differs from the predicted *timeToLive*-indexed topology-database-supplied value. Details depend on the relative locations of destination and edge-wrap stations.
b) The *purgeNow* indication, associated with topology changes, forces discards of strict-ordered traffic.

Within these code snippets, setting the *loss* variable invokes a stripping action that also updates a MIB error counter, so these abnormal conditions will not remain undetected. Within following code snippets, setting the *toss* variable invokes a stripping action with no associated error-counter update.

F.4.2 Copy decisions

Data frames are copied to clients based on *DSID*-address comparisons, as illustrated in Figure F.14. Note that early-stripped frames are not copied to the source client, if and when they reach their destination.

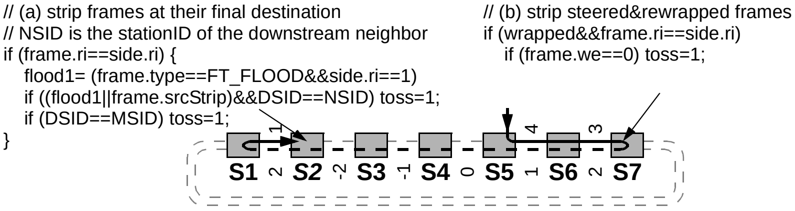


```
// copy frames to intermediate&final clients
#define FLOOD (frame.type==FT_FLOOD)
if (frame.ri==side.ri) {
    if (DSID!=MSID)
        if (frame.srcStrip||FLOOD) copy=1;
    else
        if (!(frame.srcStrip||(FLOOD&&side.ri==1))) copy=1;
}
```

Figure F.14—Strip decisions

F.4.3 Toss removals

Frames are expected to be (a) stripped when they reach their expected destination, as illustrated in Figure F.15. The destination may be the explicitly specified by the frame's implied DSID (destination station identifier), or implicity specified (b) when non-wrap traffic reaches and edge-point station.



```
// (a) strip frames at their final destination
// NSID is the stationID of the downstream neighbor
if (frame.ri==side.ri) {
    flood1= (frame.type==FT_FLOOD&&side.ri==1)
    if ((flood1||frame.srcStrip)&&DSID==NSID) toss=1;
    if (DSID==MSID) toss=1;
}
```

```
// (b) strip steered&rewrapped frames
if (wrapped&&frame.ri==side.ri)
    if (frame.we==0) toss=1;
```

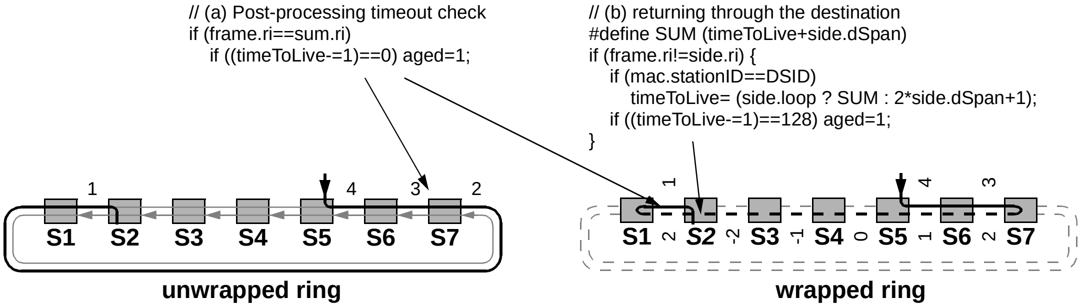
Figure F.15—Strip decisions

F.4.4 Aged frame aging

Editors' Notes (DVJ): *To be removed prior to final publication.*

This discussion on timeToLive aging is orthogonal to the discussions on flooding rules, but applies to all forms of wrapped frames. These timeToLive changes avoid potential deadlock scenarios.

For data frames, the *timeToLive* field is set to *hops* when the frame is first transmitted and is (a) decremented when passing through each station, as illustrated in Figure F.15. When the *timeToLive=0* value is reached or the originating run, the frame is discarded before transmission.



```
// (a) Post-processing timeout check
if (frame.ri==sum.ri)
    if ((timeToLive-=1)==0) aged=1;
```

```
// (b) returning through the destination
#define SUM (timeToLive+side.dSpan)
if (frame.ri!=side.ri) {
    if (mac.stationID==DSID)
        timeToLive= (side.loop ? SUM : 2*side.dSpan+1);
    if ((timeToLive-=1)==128) aged=1;
}
```

Figure F.15—Basic *timeToLive* aging

On a wrapped ring, the *timeToLive* value is (b) compensated when returning through the destination station S2, as follows:

a) Loop. If the destination database has no knowledge of the edge-generating condition, then *timeToLive* is incremented based on the loop circumference (assuming no station removals).
b) Chain. If the distance-to-edge is known, then *timeToLive* is reset based on the remaining round-trip (edge-and-back) hop-count distance.

NOTE—If the edge station is unknown (the edge-wrap information has not yet reached the destination), the revised *timeToLive* is estimated based on the apparent loop circumference. This estimate is correct if all stations are present but one link is absent, but is erroneous if one or more stations is absent. Some strict-ordered frames may therefore be lost.
NOTE—When the edge-wrap condition is known, the edge-station check correctly handles missing-link as well as missing-station protection conditions.

F.4.5 Duplicate deletion code

Duplicate deletions involves special frame processing at the wrap points, as follows:

```
// Process discovery frames received from opposing ringlet
#define DLID (fPtr->destinationAddress)                     // Destination-field address
#define DSID ((DLID&MCAST) ? fPtr->sourceStationID:DLID)    //  Effective DSID
#define SSID (fPtr->sourceStationID)
#define MSID (sPtr->stationID)
#define SPAN (sPtr->dSpan+sPtr->sSpan)
#define LEAD(hops) (hops>sPtr->dSpan&&fPtr->we==1&&fPtr->ws==0)
#define DEST(hops) (sPtr->dSids(hops))
#define FLOOD (fPtr->type==FT_FLOOD)
#define FLOOD1 (FLOOD&&sPtr->ri==1)
#define HOPS (fPtr->timeToLive)
#define SUM (HOPS+sPtr->dSpan)
#define SET (2*sPtr->dSpan+1)
int
TossDetection(SideData *sPtr, Frame *fPtr)
{   uInt8 check;
    uInt1 same, wrapped, flood1, loss=0, copy=0. toss=0, aged=0, long;

    assert(sPtr!=NULL&&fPtr!=NULL);                         // Debug consistency check
    same= (fPtr->ri==sPtr->ri);                             // Establishing short names
    wrapped= (sPtr->loop==0);                               // The station is wrapped
    if (same) {                                             // Copy to client rules
        if (sPtr->so) {                                     // Strict ordered discards
            if (dPtr->loop&&DSID!=DEST(HOPS)) loss=1;       // Passthru consistency check
            if (fPtr->we==0&&sPtr->purgeNow) loss=1;        // Reconfiguration check
        }
        if (DSID==MSID)                                     // At the destination,
            if (fPtr->srcStrip==0&&FLOOD1==0) copy=1;       //  most frames are copied
        else                                                // Before the destination,
            if (fPtr->srcStrip||FLOOD) copy= 1;             //  flooded frames are copied
        if ((fPtr->srcStrip||FLOOD1)&&DSID==NSID) toss=1;   // Redundant stripped early
        if (wrapped&&fPtr->we==0) toss=1;                   // Steered frames not wrapped
        if ((fPtr->timeToLive-=1)==0) aged=1;               // Sending timeout
    } else {
        if (DSID==MSID) {                                   // Returning by destination
            if (fPtr->we==0) loss=1;                        // Second pass is illegal
            fPtr->we= 0;                                    // Validate destination passed
            fPtr->timeToLive= sPtr->loop ? SUM:SET;         // Adjusted value selected
        }
        if (wrapped&&fPtr->we==1) loss=1;                   // Destination was missing
        if ((fPtr->timeToLive-=1)==128) aged=1;             // Returning timeout
    }
    return((loss?LOSS:0)|(copy?COPY:0)|(toss?TOSS:0)|(aged?AGED:0));
}
```

F.5 Frame forwarding

F.5.1 Local unicast forwarding

Local transfers involve prepending of leading *timeToLive* and *flags* information, to ensure reliable local delivery, as illustrated in Figure F.16. In this and following illustrations, the field that supplied the DSID (destination station identifier) addresss is identified by the italics font.

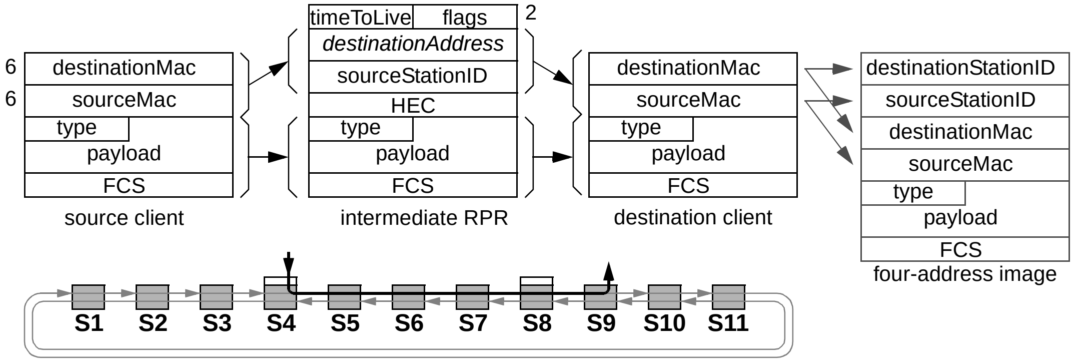


Figure F.16—Local unicast forwarding

NOTE—Each RPR frame has logical destinationStationID, sourceStationID, destinationMac, and sourceMac addresses, from the stripping rules perspective, although redundant fields are not transported. To better understand stripping-rule functionality, the equivalent four-address mappings are illustrated for each forwarding scenario.

F.5.2 Local multicast forwarding

Local transfers involve prepending of leading *timeToLive* and *flags* information, to ensure reliable RPR-local delivery, as illustrated in Figure F.16.

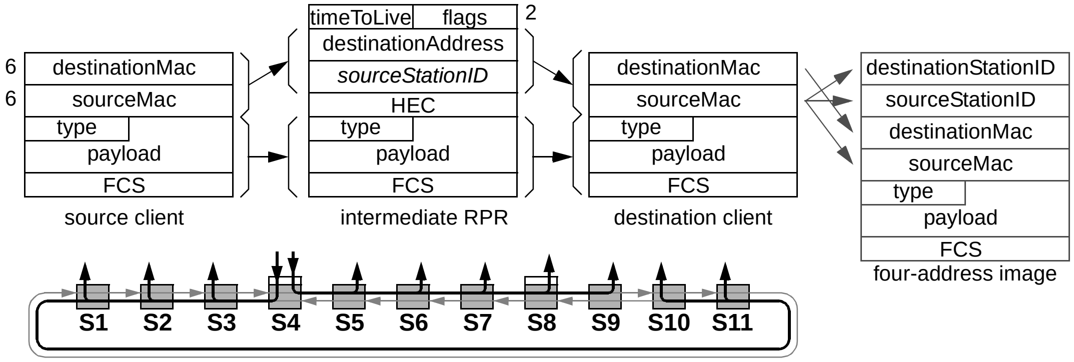


Figure F.17—Local multicast forwarding

NOTE—The multicast bit in the *destinationAddress* field implies the *DSID* is supplied by the *sourceStationID* value.

F.5.3 Remote forwarding

Remote-source or remote-destination transfers involve prepending of additional 48-bit *destinationStationID* and *sourceStationID* components to ensure reliable local delivery, as illustrated in Figure F.18.

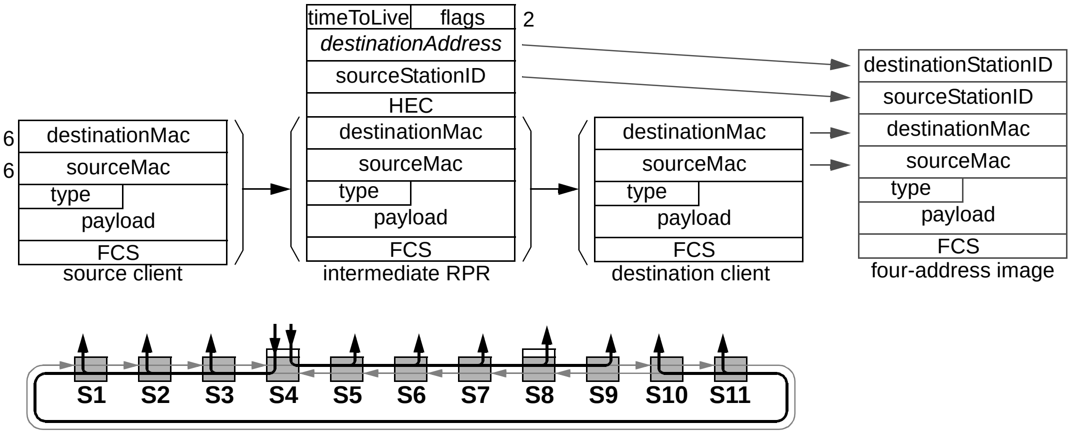


Figure F.18—Flood frame format

F.5.4 Locally sourced framing

The content of locally sourced frames can be explained in a pictorial fashion, as illustrated in Figure F.19. A change in the *srcStrip*-bit definition would compact one of the formats, a described in F.5.6.

Figure F.19—Locally sourced frames

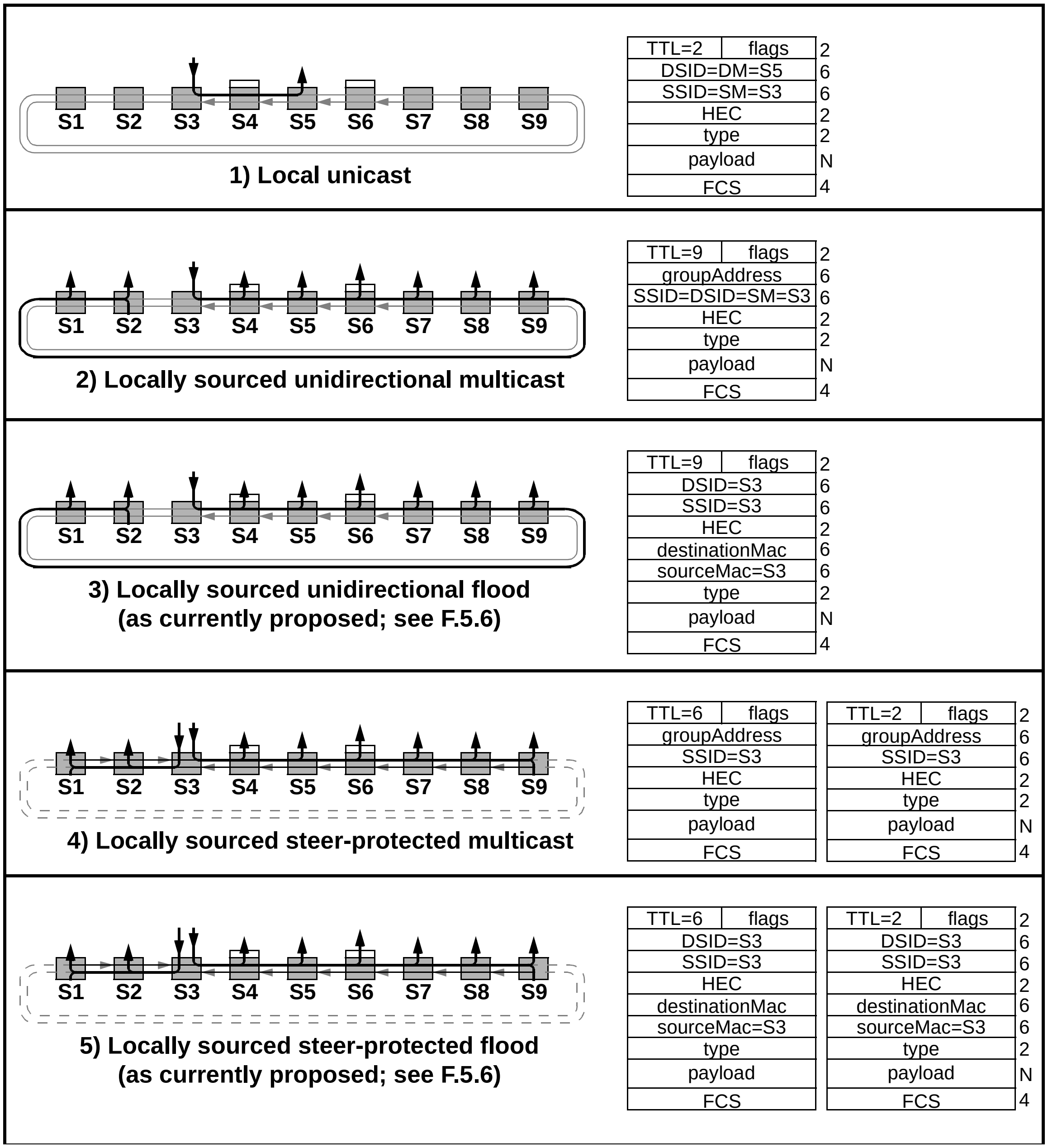
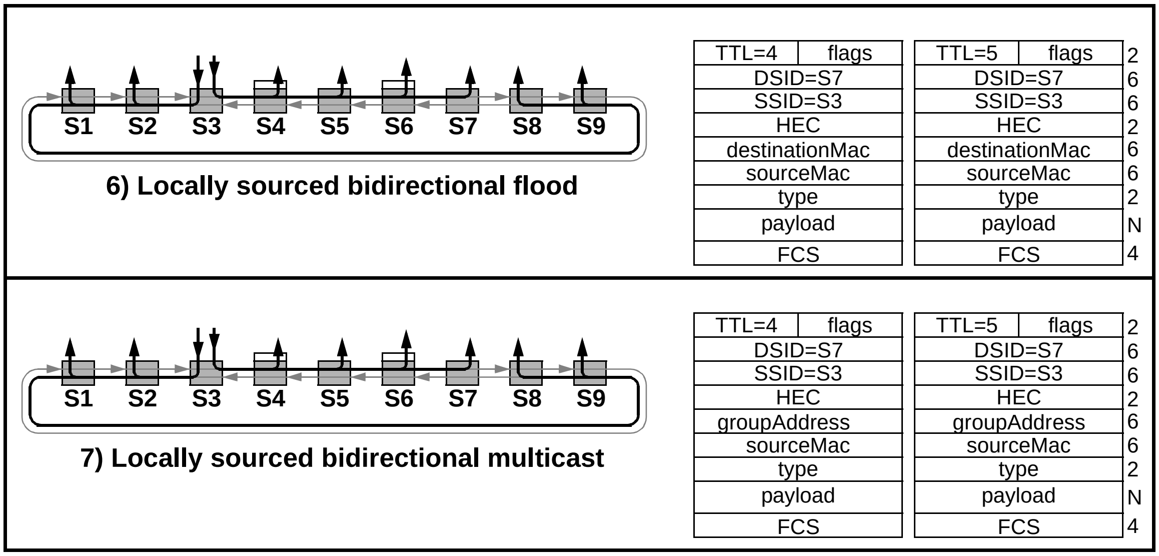


Figure F.19—Locally sourced frames



6) Locally sourced bidirectional flood

TTL=4	flags	TTL=5	flags	2
DSID=S7		DSID=S7		6
SSID=S3		SSID=S3		6
HEC		HEC		2
destinationMac		destinationMac		6
sourceMac		sourceMac		6
type		type		2
payload		payload		N
FCS		FCS		4

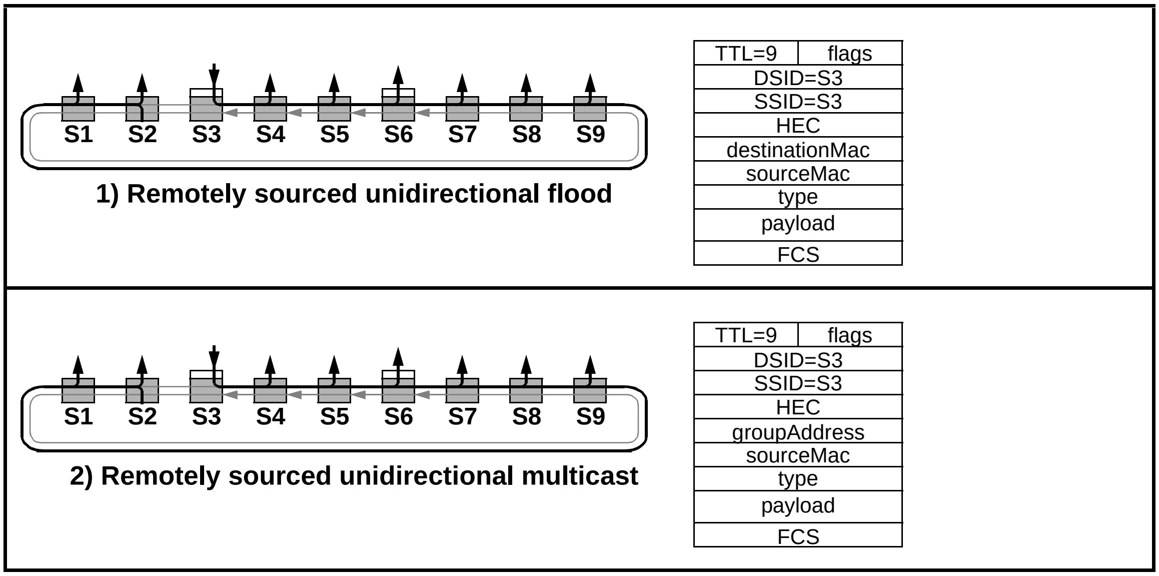
7) Locally sourced bidirectional multicast

TTL=4	flags	TTL=5	flags	2
DSID=S7		DSID=S7		6
SSID=S3		SSID=S3		6
HEC		HEC		2
groupAddress		groupAddress		6
sourceMac		sourceMac		6
type		type		2
payload		payload		N
FCS		FCS		4

F.5.5 Remotely sourced frames

The content of remotely sourced frames can be explained in a pictorial fashion, as illustrated in Figure F.20. The enhanced bridging definition would reduce remote-unicast flood scoping, a described in F.5.6.

Figure F.20—Remotely sourced frames



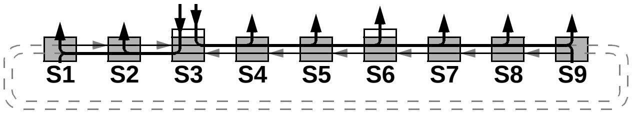
1) Remotely sourced unidirectional flood

TTL=9	flags
DSID=S3	
SSID=S3	
HEC	
destinationMac	
sourceMac	
type	
payload	
FCS	

2) Remotely sourced unidirectional multicast

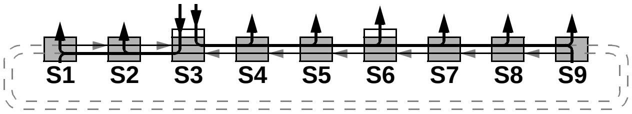
TTL=9	flags
DSID=S3	
SSID=S3	
HEC	
groupAddress	
sourceMac	
type	
payload	
FCS	

Figure F.20—Remotely sourced frames



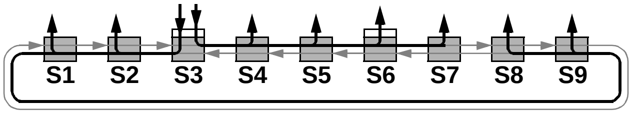
4) Remotely sourced steer-protected multicast

TTL=6	flags	TTL=2	flags	2
DSID=S3		DSID=S3		6
SSID=S3		SSID=S3		6
HEC		HEC		2
groupAddress		groupAddress		6
sourceMac		sourceMac		6
type		type		2
payload		payload		N
FCS		FCS		4



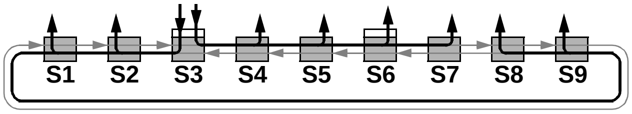
5) Remotely sourced steer-protected flood

TTL=6	flags	TTL=2	flags	2
DSID=S3		DSID=S3		6
SSID=S3		SSID=S3		6
HEC		HEC		2
destinationMac		destinationMac		6
sourceMac		sourceMac		6
type		type		2
payload		payload		N
FCS		FCS		4



3) Remotely sourced bidirectional flood

TTL=4	flags	TTL=5	flags	2
DSID=S7		DSID=S7		6
SSID=S3		SSID=S3		6
HEC		HEC		2
destinationMac		destinationMac		6
sourceMac		sourceMac		6
type		type		2
payload		payload		N
FCS		FCS		4



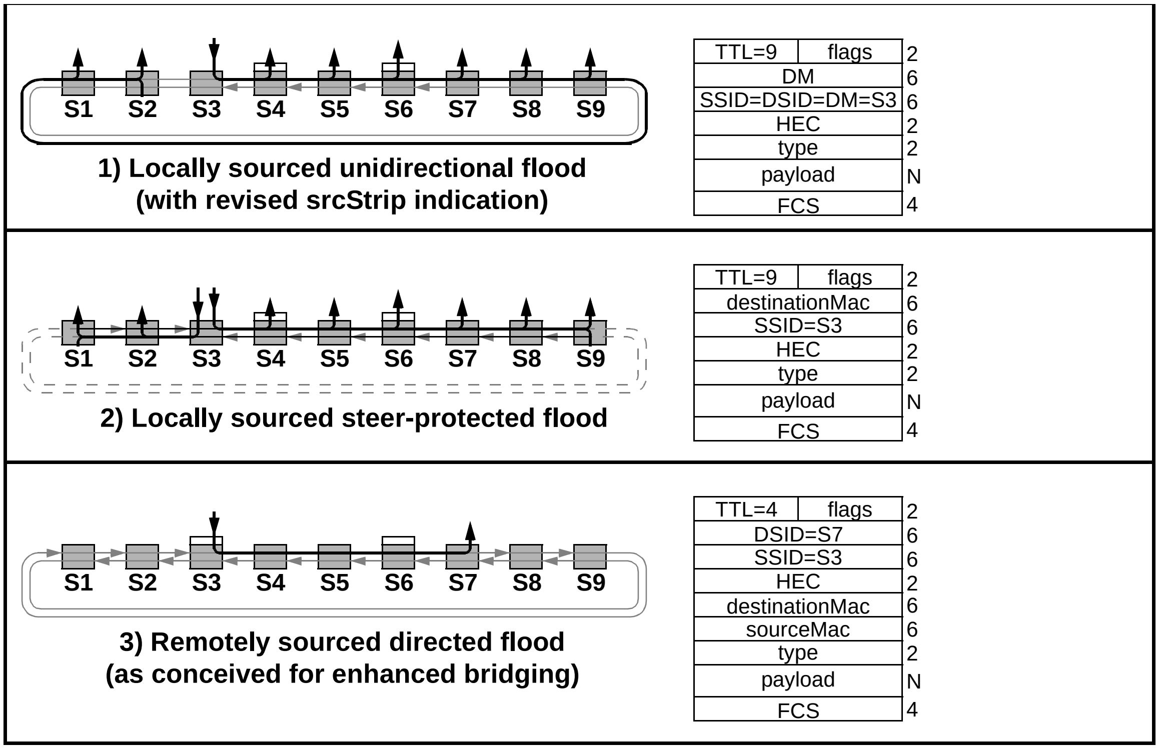
4) Remotely sourced bidirectional multicast

TTL=4	flags	TTL=5	flags	2
DSID=S7		DSID=S7		6
SSID=S3		SSID=S3		6
HEC		HEC		2
groupAddress		groupAddress		6
sourceMac		sourceMac		6
type		type		2
payload		payload		N
FCS		FCS		4

F.5.6 Other framings

The content of other frames can be explained in a pictorial fashion, as illustrated in Figure F.21. Moving the *srcStrip* bit from *destinationAddress* to the *destinationStationID* field would yield (1) a more compact frame for locally sourced unidirectional floods.

Figure F.21—Other framings



1) Locally sourced unidirectional flood
(with revised srcStrip indication)

TTL=9	flags	2
DM		6
SSID=DSID=DM=S3		6
HEC		2
type		2
payload		N
FCS		4

2) Locally sourced steer-protected flood

TTL=9	flags	2
destinationMac		6
SSID=S3		6
HEC		2
type		2
payload		N
FCS		4

3) Remotely sourced directed flood
(as conceived for enhanced bridging)

TTL=4	flags	2
DSID=S7		6
SSID=S3		6
HEC		2
destinationMac		6
sourceMac		6
type		2
payload		N
FCS		4

Enhanced bridging is expected to efficient sent bridged frames between their sourced and destination stations, eliminating unnecessary floods. However, the details of enhanced bridging are beyond the scope of this standard.

F.6 Duplicate scenarios

F.6.1 Duplicate scenarios: source&destination removals

Unidirectional flooding could be disrupted when half of the stations (including the source and destination stations) are removed, as illustrated in Figure F.22. In this example, source station *S2* along with stations *S1*, *S7*, and *S8* are removed while the *S2*-sourced frame is circulating. Correct processing involves discarding returning frames when their source is missed.

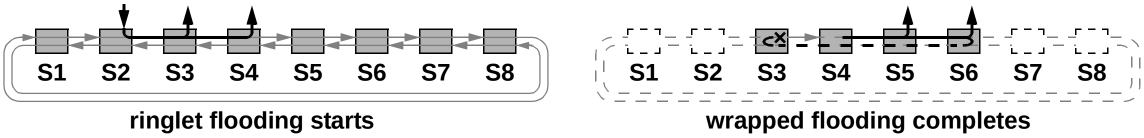


Figure F.22—Duplicate scenarios: source&destination removals

Cause: The source and destination (responsible for packet deletion) disappear while the frame circulates.
Problem: The packet may be falsely duplicated when recirculated to station *S3* and others.
Solution: Discard return-to-origin wrapped frames if the return-run destination has not been passed.

F.6.2 Duplicate scenarios: Bidirectional concurrent insertion/removal bypass

Bidirectional wrapped flooding is also susceptable to simultaneous changes in endstation locations, as illustrated in Figure F.23.

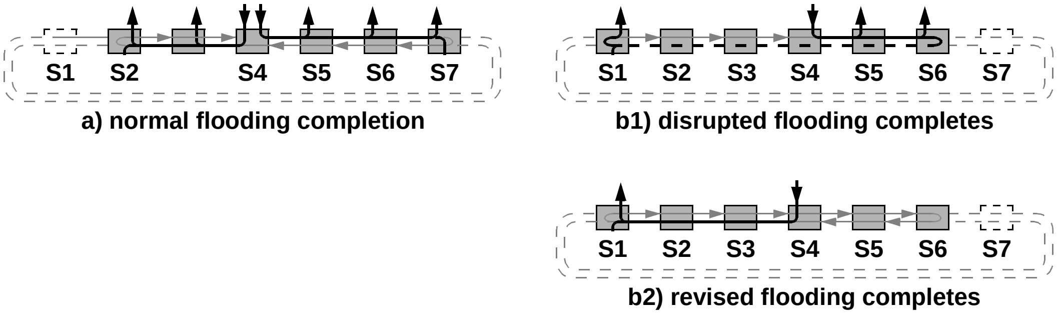

S1
S2
S4
S5
S6
S7
a) normal flooding completion
S1
S2
S3
S4
S5
S6
S7
b1) disrupted flooding completes
S1
S2
S3
S4
S5
S6
S7
b2) revised flooding completes

Figure F.23—Duplicate scenarios: Bidirectional concurrent attach/detach bypass

Details: The bidirectional flooding of (a) is disrupted by the attachment of S1 and detachment of S7. The bidirectional flooding initially goes to S2 and S7; the revised bidirectional flooding goes to S1 and S7. The initial-flood (b1) ringlet0 transmission (normally to S7) is wrapped into S1, where it is copied and removed. The revised-flood (b2) ringlet1 transmission is passed into S1, where it is copied and removed.
Problem: The packets on ringlet0 and ringlet1 are duplicated when station S1 is reached.
Solution: Discard the inconsistent *DSID!=DEST(frame.timeToLive)* frames, where the following definitions apply to this and following scenario descriptions:

```
#define DLID  frame.destinationAddress
#define DSID  ((DLID&MCAST) ? frame.sourceStationID:DLID)
#define LEAD(hops)  (hops>side.dSpan&&frame.we==1)
#define DEST(hops) (LEAD(hops) ? side.sSid(side.dSpan+side.sSpan-hops):side.dSid(hops))
```

F.6.3 Duplicate scenarios: Bidirectional concurrent insertion/removal bypass

Bidirectional wrapped flooding is also susceptable to simultaneous changes in passthrough status, as illustrated in Figure F.24.

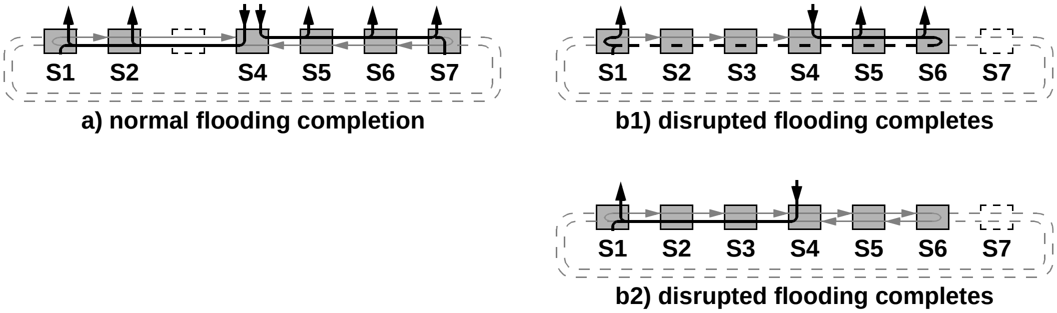

S1
S2
S4
S5
S6
S7
a) normal flooding completion
S1
S2
S3
S4
S5
S6
S7
b1) disrupted flooding completes
S1
S2
S3
S4
S5
S6
S7
b2) disrupted flooding completes

Figure F.24—Duplicate scenarios: Bidirectional concurrent insertion/removal bypass

Details: A stable bidirectional flooding topology of (a) is disrupted by the insertion of S3 and removal of S7. The (b1) ringlet0 transmission (normally to S7) is wrapped at S6, causing it to erroreously continue onward into station S1. The (b2) ringlet1 transmission passes into S2 before the S3 insertion and passes into S1 before the topology change is known to S1.
Problem: The packets on ringlet0 and ringlet1 are duplicated when station S1 is reached.
Solution: Discard inconsistent *DSID!=DEST(frame.timeToLive)* frames.

F.6.4 Duplicate scenarios: Unidirectional source bypass

Unidirectional flooding is susceptable to a source-station-pair loss during flooding, as illustrated in Figure F.25. In this example, source-station *S2* and its upstream neighbor *S3* are both bypassed while the *S2*-sourced frame is circulating. Correct source-bypass processing involves discarding the frame when its recirculates beyond its virtual source, as illustrated by the *x* marks within these Figure F.25.

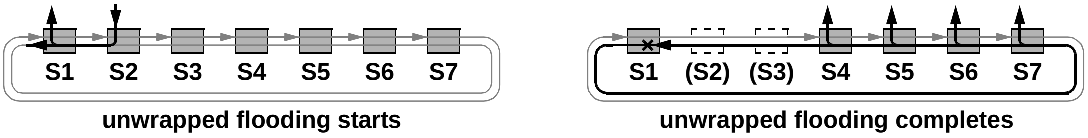
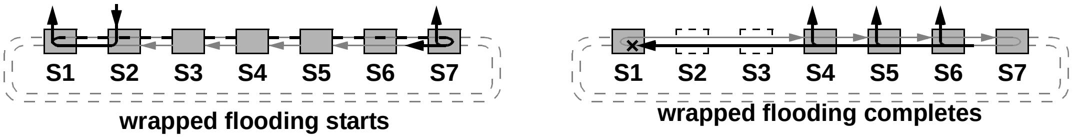


Figure F.25—Duplicate scenarios: Unidirectional source bypass

Cause: The source (that was responsible for packet deletion) disappears before its frame returns.
Problem: The packet passing through stations *S3*&*S2* may be falsely accepted by station *S1* (and others).
Solution: Discard inconsistent *DSID!=DEST(frame.timeToLive)* frames.

F.6.5 Duplicate scenarios: Unidirectional wrapped source bypass

Unidirectional wrapped flooding is also susceptable to a source-station loss during flooding, as illustrated in Figure F.24. In this example, source-station *S2* and its upstream neighbor *S3* are both bypassed while the *S2*-sourced frame is circulating on the rightside of station *S3*. Correct source-bypass processing involves discarding others' transfers when recirculate beyond the source, as illustrated by the *x* marks within these Figure F.25.



wrapped flooding starts

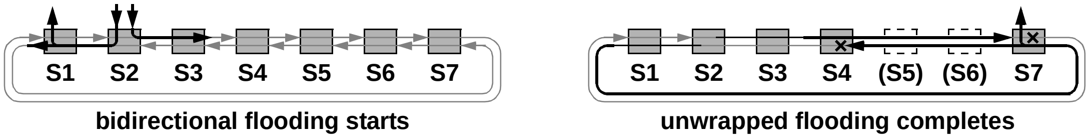
wrapped flooding completes

Figure F.26—Duplicate scenarios: Undirectional wrappes source bypass

Cause: The source (that was responsible for packet deletion) disappears before its frame returns.
Problem: The packet passing through station *S2* may be falsely accepted by station *S1* (and others).
Solution: Discard the inconsistent *DSID!=DEST(frame.timeToLive)* frames.

F.6.6 Duplicate scenarios: Unidirectional destination bypass

Bidirectional flooding is susceptable to a destination-station-pair loss during flooding, as illustrated in Figure F.25. In this example, destination stations *S5*&*S6* are bypassed while the *S2*-sourced frame is circulating. Correct destination-bypass processing involves discarding the frame when its circulates beyond its virtual destination, as illustrated by the *x* marks within these Figure F.27.



bidirectional flooding starts

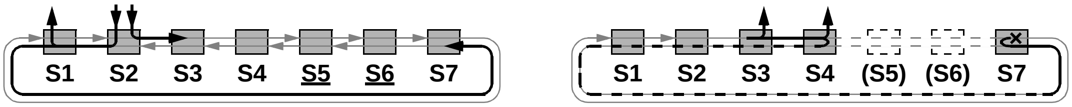
unwrapped flooding completes

Figure F.27—Duplicate scenarios: Unidirectional destination bypass

Cause: The destination (that was responsible for packet deletion) disappears before its frame arrives.
Problem: The packet passing through stations *S5*&*S6* may be falsely duplicated at station *S4, S7*, and others.
Solution: Discard the inconsistent *DSID!=DEST(frame.timeToLive)* frames.

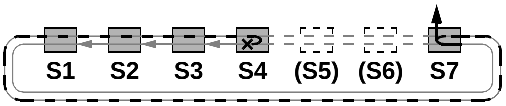
F.6.7 Duplicate scenarios: Bidirectional destination removals

Bidirectional wrapped flooding is susceptable to a destination-station-pair loss during flooding, as illustrated in Figure F.28 In this example, destination stations *S5*&*S6* are removed while the *S2*-sourced frame is circulating. Correct destination-bypass processing involves discarding the frame when its circulates beyond its virtual destination, as illustrated by the *x* marks within these Figure F.28.



bidirectional flooding starts

eastside flooding completion



westside flooding completion

Figure F.28—Duplicate scenarios: Bidirectional destination removals

Cause: The destination (that was responsible for packet deletion) disappears before its frame arrives.
Problem: The packet wrapped before stations *S5*&*S6* may be falsely duplicated at station *S4*, *S7*, and others.
Solution: Discard the inconsistent *DSID!=DEST(frame.timeToLive)* frames.