```
type graph = bool [n,n];
type kind = (PROBE, ECHO);
chan probe_echo[n](kind k; int sender; graph topology);
chan finalecho(graph topology);

process Node[p = 0 to n-1] {
  bool links[n] = neighbors of node p;
  graph newtop, localtop = ([n*n] false);
  int first, sender; kind k;
  int need_echo = number of neighbors - 1;
  localtop[p,0:n-1] = links;   # initially my links

  receive probe_echo[p](k, first, newtop);  # get probe
  # send probe on to to all other neighbors
  for [q = 0 to n-1 st (links[q] and q != first)]
    send probe_echo[q](PROBE, p, ∅);

  while (need_echo > 0) {
    # receive echoes or redundant probes from neighbors
    receive probe_echo[p](k, sender, newtop);
    if (k == PROBE)
      send probe_echo[sender](ECHO, p, ∅);
    else  # k == ECHO {
      localtop = localtop or newtop;  # logical or
      need_echo = need_echo-1;
    }
  }
  if (p == S)
    send finalecho(localtop);
  else
    send probe_echo[first](ECHO, p, localtop);
}

process Initiator {
  graph topology;   # network topology
  send probe_echo[source](PROBE, source, ∅);
  receive finalecho(topology);
}
```

Figure 9.12 Probe/echo algorithm for computing the topology of a graph.

Copyright © 2000 by Addison Wesley Longman, Inc.