original multicast strategy

```
// loop

for (const auto& nexthop : nexthops) {
    Face& outFace = nexthop.getFace();  // nexthop's face
    RetxSuppressionResult suppressResult =
m_retxSuppression.decidePerUpstream(*pitEntry, outFace);

    if (suppressResult == RetxSuppressionResult::SUPPRESS) {    // suppress
        NFD_LOG_DEBUG(interest << " from=" << inFace.getId()
                << "to=" << outFace.getId() << " suppressed");
    isSuppressed = true;
    continue;
    }

    // in = out / linktype != ad-hoc
    if ((outFace.getId() == inFace.getId() && outFace.getLinkType() !=
ndn::nfd::LINK_TYPE_AD_HOC) ||
    wouldViolateScope(inFace, interest, outFace)) {
    continue;
    }

    //cout << "FaceID : " << outFace.getId() << endl;

    this->sendInterest(pitEntry, outFace, interest);    // send multicast
    NFD_LOG_DEBUG(interest << " from=" << inFace.getId()
                    << " pitEntry-to=" << outFace.getId());

    if (suppressResult == RetxSuppressionResult::FORWARD) {     // forward
        m_retxSuppression.incrementIntervalForOutRecord(*pitEntry-
>getOutRecord(outFace));
    }
    ++nEligibleNextHops;
}

if (nEligibleNextHops == 0 && !isSuppressed) {
    //cout << "numOfNextHop : " << numOfNextHop << endl;
    NFD_LOG_DEBUG(interest << " from=" << inFace.getId() << " noNextHop");

    lp::NackHeader nackHeader;
    nackHeader.setReason(lp::NackReason::NO_ROUTE);
    this->sendNack(pitEntry, inFace, nackHeader);

    this->rejectPendingInterest(pitEntry);

}
```

INFORM strategy : choice best & random then send

```
// best choice
auto it = nexthops.end();

it = std::find_if(nexthops.begin(), nexthops.end(), [&] (const auto& b_nexthop) {
    return isNextHopEligible(inFace, interest, b_nexthop, pitEntry);
});

// random choice
if (!hasFaceForForwarding(inFace, nexthops, pitEntry)) {
    this->rejectPendingInterest(pitEntry);  // reject
    return;
}

fib::NextHopList::const_iterator selected;

do {  // choice algorithm

    boost::random::uniform_int_distribution<> dist(0, nexthops.size() - 1);
    const size_t randomIndex = dist(m_randomGenerator);

    uint64_t currentIndex = 0;

    for (selected = nexthops.begin(); selected != nexthops.end() && currentIndex
!= randomIndex;
        ++selected, ++currentIndex) { }

} while (!canForwardToNextHop(inFace, pitEntry, *selected));

// loop

for (const auto& nexthop : nexthops) {
    Face& outFace = nexthop.getFace();  // nexthop's face
    RetxSuppressionResult suppressResult =
m_retxSuppression.decidePerUpstream(*pitEntry, outFace);

    if (suppressResult == RetxSuppressionResult::SUPPRESS) {    // suppress
    NFD_LOG_DEBUG(interest << " from=" << inFace.getId()
                << "to=" << outFace.getId() << " suppressed");
    isSuppressed = true;
    continue;
    }

    // in = out / linktype != ad-hoc
    if ((outFace.getId() == inFace.getId() && outFace.getLinkType() !=
ndn::nfd::LINK_TYPE_AD_HOC) ||
        wouldViolateScope(inFace, interest, outFace)) {
    continue;
    }
```

```
    // send best&random selected
    if (outFace.getId() == it->getFace().getId() || outFace.getId() == selected-
>getFace().getId()) {

        cout << "Route : " << inFace.getId() << " -> " << outFace.getId() << endl;

        this->sendInterest(pitEntry, outFace, interest);    // send multicast

        NFD_LOG_DEBUG(interest << " from=" << inFace.getId()
                              << " pitEntry-to=" << outFace.getId());
    }

    if (suppressResult == RetxSuppressionResult::FORWARD) {     // forward
    m_retxSuppression.incrementIntervalForOutRecord(*pitEntry-
>getOutRecord(outFace));
    }
    ++nEligibleNextHops;
}

if (nEligibleNextHops == 0 && !isSuppressed) {
    NFD_LOG_DEBUG(interest << " from=" << inFace.getId() << " noNextHop");

    lp::NackHeader nackHeader;
    nackHeader.setReason(lp::NackReason::NO_ROUTE);
    this->sendNack(pitEntry, inFace, nackHeader);

    this->rejectPendingInterest(pitEntry);
}
```