Congestion Control – Design Specifics

1. Congestion Detection

At each router: Use the estimator and setpoint of CoDel (https://datatracker.ietf.org/doc/draft-ietf-aqm-codel/), which records the "packet sojourn time".

In the following, we will use the term **queue size** to refer to the time that the sojourn time remained above the target value (default: 5 ms). We will consider a link as "congested" when this time is above the interval (default: 100 ms).

We do not use CoDel's control loop (spacing out packet drops), but instead implement our own.

2. Congestion Signaling

At each router, on each incoming data packet:

1. Check the packet tag contains a queue size entry. Call it *tagQueueSize*
2. For each downstream face (PIT entry where the data packet will be sent):
    - Look up its own queue size, called *localQueueSize*.
    - If there is an error (e.g., the router doesn't use the codel AQM), set localQueueSize = -1;
    - replace the packet tag with MAX(tagQueueSize, localQueueSize) and send it to the downstream.

3. Reaction at the Consumer

Implement a TCP RENO like behavior that uses "slow start" and congestion avoidance. The value "sstresh" starts out as sufficiently high.

On each incoming data packet:

```
If (tagQueueSize > 0):
    If (NOT markedRecently):
        sstresh = cwnd/2.0;
        cwnd = sstresh;
        lastMarkedSeq = thisSeq;
    Else:
        cwnd -= 1.0/cwnd;
Else:
    If (cwnd <= sstresh):
        cwnd++;
    Else:
        cwnd += 1.0/cwnd;
End
```

Explanation of the pseudocode: For the first marked packet, the consumer performs a multiplicative decrease. Then for a time "markedRecently" (default: 1.1 * RTT), the consumer prohibits multiplicative decreases on marked packets and instead performs a linear decrease (alternative: keep cwnd the same).

For each unmarked packet, the consumer either performs an exponential increase (when in the slow start phase), or an additive increase (after the slow start phase).

When receiving a timeout set sstresh=cwnd/2.0 and cwnd=1;

When receiving a congestion NACK (indicating that the requested packet was dropped), the behavior is the same as on receiving a timeout.

The consumer should set its timer to a value high enough to avoid false positive timeouts (e.g. 5 seconds). Timers can be set rather too high than too low, because when the network is operating properly, timeouts *should not occur* (NACKs should also be rare). Moreover, thanks to the packet marks, timeouts are no longer needed to adjust the congestion window.

The router buffers inside the network should be chosen large enough to absorb any temporary traffic bursts and, in the worst case, send an explicit NACK instead of silently dropping packets.

4. Multipath Forwarding Strategy

4.1. Simple Design

For each FIB prefix: Keep a `map<FaceId, bool disabled>` that indicates whether the current interface is disabled due to congestion on the link.

On each incoming data packet:

```
If (tagQueueSize > 0):
    disableFace(Prefix, incomingFaceId)
Else:
    enableFace(Prefix, incomingFaceId)
End
```

On each incoming interest packet:

- Create a list of eligible faces (ones that are not deactivated)
- Forward packet randomly on one of the faces (equal forwarding probability)
- If all faces are disabled: Choose face randomly among disabled faces (equal probability)

The problem of this simple forwarding strategy is that there is a latency of one link propagation delay between the upstream router marking a face as congested and the downstream router setting its state to congested (or vice versa with removing the congestion marks). Completely avoiding one interface during that time tends to underutilize the link. Thus, the following forwarding strategies change the forwarding ratio more slowly.

4.2. Carofiglio et al's Forwarding Strategy

Here we adjust the forwarding ratio in order the equalize the number of outstanding interests, as suggested in http://www.inf.usi.ch/phd/papalini/pdf/icnp13.pdf and implemented here: http://systemx.enst.fr/archives/NDN-0.2.0-custom.tgz

As Nguyen et al. (http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7247397&tag=1) pointed out, "PI-based forwarding prefers routes with narrower available bandwidth which is counter-intuitive if we consider throughput maximization as the objective."

In my measurements, the PI-based distribution could not achieve the optimal combined link bandwidth on heterogeneous paths: If the paths had a link bandwidth ratio of 75:25, the strategy split traffic only to about 60:40.

Disclaimer: The non-optimal split ratio may be caused by my re-implementation (I ported their forwarding strategy to ndnSIM), rather than an inherent design flaw.

4.3 Main Forwarding Strategy Design

Per FIB Prefix:

- Keep a `map<FaceId, double forwPerc>` forwPercMap for the forwarding percentages of each interface. Initialize this map to either an equal split, or 100% for the best interface (e.g. shortest path) and 0% for all others.
- Keep a `map<FaceId, Markings>` markingPercMap, where "Markings" are a list of boolean values, with a function to compute the moving average (between 0 and 1).

On each incoming data packet:

```
If (tagQueueSize > 0):
    markPercMap.add(inFaceId, true)
Else:
    markPercMap.add(inFaceId, false)
End
```

Every forwarding update interval (default: 20 ms - 200 ms):

- Reduce forwarding percentage of the face with the most markings (default: by 1% to 3%)
- Increase forwarding percentage of all other faces equally (so the sum remains 100%)
- clear markings of all faces.

On each incoming interest:

- Forward packets probabilistically based on forwPercMap.

Extension: Suppress Marks State When starting with a suboptimal forwarding distribution, the consumer might receive marks (thus reduce its sending rate) even though a better alternative link is available at one router. To prevent an unnecessary window decrease, we introduce a "suppress marks" state, where incoming marks are not relayed further downstream if a better path is available.

During the forwarding update procedure (each fw update interval):

```
If there exists an eligible face without 0% marks during the last interval:
    set supprMarks = true;
End
```

On incoming data:

```
If (supprMarks == true):
    // Ignore queue size of tag, but not local queue size
    tagQueueSize = 0;
End
```

Problem: Adapting the forwarding ratio is quite slow (can be in the order of seconds until the optimal ratio is achieved). This keeps the consumer too long in the “slow start” phase, thus causes an unnecessary amount of interest in router queues.

Solution: ?

Draft 1 – Klaus Schneider