What if PIT is Full?

Junxiao Shi, 20170303

# Scenario

The pending Interest table on a router has limited capacity. PIT capacity is limited by available memory capacity and memory bandwidth (to achieve line speed forwarding, CPU can only read a limited amount of memory within time allocated for processing a packet). When the PIT is full, the router would not be able to accept new Interests without first deleting some PIT entries. When this happens, there would be a choice between rejecting new Interests and evicting existing PIT entries. In addition, when the PIT is almost full, the router can start signaling downstream about this situation, so that a cooperative downstream can decrease its Interest rate and prevent filling the PIT.

This document investigates the PIT capacity problem with the following topology:

A---  
 \  
B----R----P  
 /  
C---

The router R has a limited PIT capacity of 300 entries. Downstream A, B, and C send Interests toward upstream P. P does not send Interests. There is no link congestion on downstream side.

# PIT capacity as a flow control problem

PIT capacity limit can be modeled as a flow control problem, in which the PIT space is analogous to a receive buffer. With sliding window protocol, R can inform each downstream how much PIT space it is allowed to use. For example, R tells A that it will only keep 100 PIT entries; afterwards, if A has sent 100 Interests, it would stop sending more Interests until some of them have been satisfied or expired.

A challenge in this model is: how many PIT entries should R allow each downstream to use? Recall that R has a PIT capacity of 300 entries, and it serves 3 downstream nodes. If R allows each consumer to use 100 PIT entries, it would not never run out of PIT space, but network resources are not effectively utilized if only one downstream is actively sending Interests. On the other hand, if R allows each downstream to use all 300 PIT entries, a malicious downstream could fill up the PIT, preventing other consumers from communicating.

# Max-min fairness

Max-min fairness is a common method for allocating bandwidth resource. Suppose A, B, and C all want to send IP packets to P at rates 4Mbps, 15Mbps, 25Mbps, but R-P link has only 30Mbps bandwidth. With max-min fairness, each of A, B, C is initially allocated 10Mbps bandwidth. Then, since A is sending less than 10Mbps, the remaining 26Mbps is equally divided among B and C. Eventually, the sending rates are 5Mbps, 13Mbps, 13Mbps. Max-min fairness reacts well to change of demand. Suppose A now wants to send 8Mbps instead of 4Mbps, the sending rates of B and C can quickly be reduced to 11Mbps each.

In reality, it is possible to assign a weight to each downstream. For example, if downstream C is aggregating traffic from many end hosts, it may be entitled to a larger share than A and B.

## Difference between IP bandwidth sharing and PIT space sharing

While max-min fairness works well with bandwidth allocation, it is not directly applicable to the PIT capacity problem, because PIT entries must be kept until they are satisfied or expired. In case of a change of demand, the router would not be able to reallocate PIT space immediately. For example, A is occupying all 300 PIT entries, and B starts sending Interests. R would not be able to accept B’s Interests without deleting some existing PIT entries.

Evicting existing PIT entries is considered a bad idea, because these Interests are already forwarded to P, which means R has committed the bandwidth for returning Data from P. Evicting existing PIT entries would not prevent P from returning requested Data (unless we add additional signaling to cancel the Interests). While R could accept B’s Interests after evicting PIT entries, the extra returning Data might cause congestion on P-R link.

Rejecting all new Interests is undesirable, too. Downstream A can maliciously send many Interests with long InterestLifetime, occupying the PIT space for a long time. In IP bandwidth sharing, A’s packets are forwarded out and forgotten quickly, leaving room in the receive buffer for B and C’s packets to compete for; in NDN’s PIT, A’s Interests stay in the PIT for a significant amount of time defined by InterestLifetime, not giving B and C any chance to compete for PIT space. Imposing a limit on maximum InterestLifetime offers limited help to mitigate this attack, because A knows when its Interests would expire and can replenish its pending Interests in time.

## Solution: reserved PIT space

A proposed solution is to reserve a small number of PIT entries for each downstream, and use max-min fairness on the unreserved PIT space.

In this solution, the PIT space is divided into a small reserved portion, and a large unreserved portion. Max-min fairness allocates the unreserved portion to active downstream nodes; the reserved portion acts like a queue, and can only be used by downstream nodes whose PIT usage is less than the fair share.

queue|unreserved-space  
 |AAAAAAAAAAAAAAAAAAAA A occupies all unreserved space  
BB |AAAAAAAAAAAAAAAAAAAA B starts  
BB |BBAAAAAAAAAAAAAAAAAA A’s Interests satisfied/expired  
BB |BBBBAAAAAAAAAAAAAAAA  
BB |BBBBBBAAAAAAAAAAAAAA  
BB |BBBBBBBBAAAAAAAAAAAA  
 |BBBBBBBBBBAAAAAAAAAA B reaches fair share and cannot use queue

For example, we divide the 300-entry PIT capacity on R to 290 entries of unreserved space, and 10 entries of reserved queue. Initially, only downstream A is active, and it can use up to 290 PIT entries. When downstream B becomes active, R would accept 10 Interests from B, placing them into the queue. A’s fair share is reduced to 280 although it is still using 290 entries. Once 10 of A’s PIT entries are satisfied or expired, B’s PIT entries are moved from the queue to the unreserved space. The queue is empty again, but only B and C can use it. If B has more Interests to send, they would go into the queue and further reduce A’s fair share. Eventually, A and B can each use 145 PIT entries in the unreserved space, and none in the queue.

For this to be effective, it is necessary to impose a maximum PIT entry lifetime limit. Otherwise, if malicious downstream A sends Interests with long InterestLifetime, there would be a significant duration in which A’s PIT usage (290) is higher than its fair share (280), while B occupies all space in the queue, denying C’s access to the PIT space. With a PIT lifetime limit of 1 second, A’s PIT usage can be reduced by 10 entries in every 1 second.

## Queue space allocation

In the above solution, any downstream whose PIT usage is lower than the fair share is entitled to use the queue. A drawback of this allocation method is that, when B occupies all space in the queue, C would not be able to start.

An alternative method is to reserve a separate queue for each consumer. For example, each of A, B, C has a 2-entry queue, and the rest 294 entries are unreserved. Initially, A is able to use all 294 entries in the unreserved space. When B becomes active, R would accept 2 Interests from B and reduce A’s fair share in the unreserved space to 292. Compared to a single queue, this method allows all inactive downstream nodes to start simultaneously, at the cost of longer convergence time because now A’s PIT usage can only be reduced by 2 entries in every 1 second.

# PIT full signaling

R can inform A, B, and C their PIT space allocation via NDNLP. The signaling does not have to include exact numbers. Instead, a protocol similar to Random-Early-Drop (RED) can be utilized.

The RED protocol works as follows. When PIT usage is below low threshold, every Interest is accepted. When PIT usage is between low threshold and high threshold, Interests are still accepted, but a congestion signal is returned, asking the downstream to slow down. When PIT usage is above high threshold, incoming Interests not only trigger the congestion signal, but also are probabilistically dropped based on its InterestLifetime and PIT occupancy level.

# Relation to PIT entry lifetime

This is closely related to PIT entry lifetime (hold time). When A is occupying most of PIT space and B is starting, short hold time allows R to quickly remove A's PIT entries, so that B can ramp up its sending rate faster.

To ensure Data returned to R is useful (not unsolicited), the InterestLifetime from R to P should be no more than the hold time. Therefore, adopting a short hold time requires resending Interests more often, not only between A and R, but also between R and P. There would be CPU overhead on every node for resending and processing those Interests (link congestion is minor because Interest packets are small). When A is the only active downstream, this would be unnecessary overhead.

Adopting a long hold time has the opposite effect: Interest resending occurs less, reducing CPU overhead, but harms fairness as B needs to wait longer to achieve its fair rate.

This analysis indicates that a fixed hold time is undesirable. Instead, the hold time should be dynamically adjusted according to PIT occupancy level. Since A's fair share is at least 100 entries, R could hold A's first 100 PIT entries indefinitely (hold time equals incoming InterestLifetime). Then, R starts to place restriction on hold time of A's PIT entries when its PIT usage is greater than 100 entries. For example, the maximum hold time for A's 101st PIT entry can be set to 10 seconds, and it linearly reduces to zero when A is using all unreserved PIT space. However, if the hold time would be less than round trip time, R should stop accepting A's Interests altogether because there is no way to retrieve Data within hold time.

Signaling of current hold time can be piggybacked onto link layer packets. To reduce signaling overhead, router R should use discrete steps of hold time values rather than adjusting at every incoming Interest. Hold time signaling does not replace link-layer acknowledgement and is not network-layer acknowledgement, so that it does not come with a sequence number or Interest digest. Therefore, R must account for the delay between R adjusting hold time and A receiving new hold time, and be slightly relax on the actual hold time (i.e. the actual hold time should be greater than the hold time told to A).

# Per-flow, per-neighbor, or per-consumer fairness

Common problems of IP per-flow max-min fairness include: (1) any user can create multiple flows in order to get more bandwidth allocation (2) there is no memory of previous bandwidth allocation, so that the scheme unfairly benefits constant background traffic and punishes short-term high-bandwidth traffic.

In NDN, there is no well-defined concept of “flow”. Per-name-prefix fairness is the closest to per-flow fairness, and it suffers from the same weakness as in IP: a consumer can send Interests under multiple different name prefixes, in order to get more network resources.

Per-neighbor fairness is achievable through max-min fairness with reserved PIT space, as described above. It suffers from the problem of unfairly benefiting constant background traffic.

Arguably, per-consumer fairness may be more desirable than per-neighbor fairness. In per-neighbor fairness, neighbor B may represent 1000 consumers behind it, while neighbor C has only 1 consumer. Enforcing equality of PIT and bandwidth resources for neighbors leads to a highly unequal share of resources for consumers.

On the other hand, a router in the middle of the Internet has no incentive to optimize for a remote end user, unless such an incentive has been passed to this router. By default, data retrieval is hop-by-hop: a downstream node sends an Interest to an upstream, and pays for PIT and bandwidth resources. If neighbor B has 1000 consumers behind it, it should pay more to R, earn a larger weight, so as to get more PIT space allocation.

# RED with different drop probability

Random Early-Drop can be used to prevent a router’s PIT from filling up, and allow every downstream an opportunity to compete for PIT space, but there is no fairness: whoever sending Interests the fastest gets most PIT space. However, we can modify RED to add fairness: when the PIT is in danger of being filled up, packets from a downstream that occupies more PIT space has a higher probability of being dropped.

A drawback of this scheme is that there is no incentive for a downstream to use reasonably small InterestLifetime. Since A and B share a router R, if A is sending Interests with large InterestLifetime and R forwards them as is, P would punish R for occupying too much PIT space, and indirectly B gets punished even if it's not misbehaving.

Notes (by Beichuan):

Concepts:

* L = Interest Lifetime, set by applications, indicating when the app still wants the data.
* H = PIT hold time, set by NFD, how long the NFD will keep this interest in the PIT.
* E = Explore time, set by NFD, for how long the NFD will keep trying alternative faces to get the data.

Issues:

* Manage PIT occupancy, i.e., prevent it from overflowing.
* Allow flexible strategy decisions about when and where to try to get data.
* Encourage setting L appropriately.
* Loop detection

Analysis:

* L > H > E ?
* If H is too short, it may not be able to detect loops.
  + Solution: enforce an Hmin, or Dead Nonce List.
* If H is too long, it may fill up the PIT.
* S(PIT) = Size(Interest)\*(RTT\*BW/Size(Data))
  + Assuming RTT=1s, BW = 10Gbps, Data = 1000B, Interest = 100B, then PIT is 1Gb. Not a problem for regular PCs.
  + For small devices with WiFi interface, say 10Mbps, PIT is 1Mb. Still manageable, right?
  + Is the PIT size a problem that we need to worry now?

Existing Implementations:

* CCNX 0.8 strategy:
  + H = L
  + E: try every nexthop, then stop, even when more interests are coming, i.e., not supporting retransmission, which will not be a problem if assuming a reliable Link Protocol.
* Cheng’s strategy:
  + H = L
  + E =~ RTO. During this time, continue try different nexthop after getting NACK, stop after RTO. Resume after receiving a retransmission.
* NFD’s best-route strategy:
  + H = L
  + E: round-robin among nexthops, triggered by downstream retransmissions, with exponentially increasing timers?
* Van’s refresh idea:
  + H = min(L, Hmax)
  + E = H
  + It’s downstream’s responsibility to retransmit interest to refresh the PIT entry at the upstream so that it will not be removed if the downstream still wants the data.
  + Downstream learns Hmax from routing announcements, either in regular routing protocol, or self-learning attachment.
* Our proposal:
  + H = L
  + When PIT occupancy is higher than a threshold, send a congestion signal to downstream. If it’s too high, we can drop an incoming interest by a probability that is based on the PIT occupancy and the interest’s Lifetime, e.g., the longer the lifetime, the more likely to be dropped.