Performance Evaluation of In-network Packet Retransmissions using Markov Chains

Runa Barik, Michael Welzl, Peyman Teymoori, Safiqul Islam, Stein Gjessing
Networks and Distributed Systems Group
Department of Informatics, University of Oslo
{runabk, michawe, peymant, safiquli, steing}@ifi.uio.no

***Abstract*—We consider the retransmission of packets inside the network on a segment of an end-to-end path. Using a Markov chain formulation of the problem, we evaluate the effect of the loss probability in the network segment that is responsible for retransmission as well as the ratio between the local RTT and the end-to-end timeout. We also obtain the optimal cache size required for retransmission depending on the packet loss and cache blocking probabilities. Our study reveals that the local RTT of the path segment for retransmission as a function of the end-to-end timeout significantly influences the packet caching time. We also observe that a small increment in the packet loss probability above a threshold can severely affect the percentage of cache filling irrespective of the cache size. Use cases to which our findings apply include a recent IETF proposal called "LOOPS", recursive network architectures and proposals that use hop-by-hop retransmissions in special network scenarios, e.g. for Information-Centric Networking (ICN).**

***Index Terms*—Packet Retransmission, LOOPS, Markov Chain, RINA**

I. Introduction

The famous paper that introduced the "end-to-end argument" [1]—a guideline which "suggests that functions placed at low levels of a system may be redundant or of little value when compared with the cost of providing them at that low level"—begins with an example of a reliable file transfer. This example concludes with the finding that reliability must be carried out end-to-end in any case, and that it is therefore pointless to try to implement it inside the network.

The end-to-end argument is one of the cornerstones of the Internet's design; strictly avoiding to place application-specific functions inside the network has ensured scalability, allowing for growth at an unprecedented rate. The Internet's success, and its designers' reliance on the end-to-end argument, may well be the reason why the possibility to retransmit packets *inside* the network has not received much attention since the early 1980's (as stated in [1], in-network reliability was a common proposal at that time). We posit that it is now worthwhile to give such retransmissions a new consideration.

We do not contradict the argument: *guaranteed* reliability must be taken care of end-to-end. However, when packets are frequently dropped on path segments with a very short round-trip time (RTT), and when the end-to-end RTT is very long, retransmissions from on-path relays do help—they can get packets to the receiver faster, reducing latency.

This benefit is well known to designers of wireless network interfaces, which often carry out a form of link layer Automatic Repeat Request (ARQ)—local retransmissions that reduce the packet loss probability for the end-to-end control loop. For example, 802.11 Access Points (APs) usually have configurable retry limits to specify how often the AP should try to re-send a frame in case of failure; 7 is a common default value for short frames.

Fixed constants in APs and other link layer equipment are of course not ideal; they are a result of the separate development of Internet standards on one hand, and link layer standards on the other. Also, APs carry out two functions, not one, as part of their ARQ behavior: i) they retransmit lost frames for a certain maximum number of times, and ii) they delay all other incoming data during the retransmission procedure, thereby ensuring in-sequence delivery. The latter decision is perhaps inevitable when multiple frames make up an IP packet—however, whenever there is a one-to-one correspondence between Internet packets and link layer frames, delaying data for the sake of ordering is in conflict with the end-to-end argument because applications are better placed to decide whether they really need such ordering. Indeed, delaying frames for the sake of ordering can be detrimental when it is done for frame aggregates in case of 802.11n [2].

In this paper, we consider *only* retransmission, not ordering at the expense of delay,[1] and we ask: given the end-to-end RTT, the RTT on a path segment, and the packet loss ratios of the various parts of an end-to-end path, how long should a system ideally store packets for retries? How large is the potential benefit from in-network retransmission, depending on the above factors? Our contributions are:

- We introduce a stochastic model that describes the relationship between packet drop probabilities in different path segments, RTT, and buffer size.
- We investigate the influence of a local loss recovery mechanism: caching packets in an intermediate node and loss detection before packets being arrived at the destination using the above model. Moreover, we derive the impact of each parameter on the system behavior.

[1]We make this decision to keep our model simple, but it could obviously be extended to also capture ordering. In practice, out-of-order packets can be problematic for TCP, which generally assumes that a re-ordering degree of 3 packets or more indicates congestion. Recent TCP versions have however introduced spurious loss detection and recovery mechanisms such as Eifel [3], [4] and F-RTO [5] which enable a TCP sender to undo the congestion reaction in case it was found to be a misunderstanding. RACK [6], one of the latest additions to TCP, lets TCP interpret time instead of sequence numbers, which generally makes it more robust against re-ordering.

As we will discuss in the next section, some recent developments provide use cases for such in-network retransmissions, for which it makes sense to ask these questions. After a discussion of background work on in-network retransmission and queuing models for related work in Section II, we develop a mathematical formulation of our proposed system and describe the results in Section III. In Section IV, we evaluate the performance impact at end-systems. Section V summarizes our findings.

II. Background and Related Work

The scenario that we consider is shown in Figure 1: a sender S emits packets with rate λ towards a receiver R, which acknowledges their reception; acknowledgments (ACKs) arrive at the sender after rtt_s. If packets are not acknowledged in time, they are retransmitted after a retransmission timeout rto_s. Some real-life systems may use negative acknowledgments (NACKs) too, but because they can be lost just like data packets, NACKs must be combined with positive ACKs and a timeout. Because that only makes them an optional mechanism that can make the sender react earlier, we do not consider NACKs for the sake of simplicity.

On their way towards the receiver, packets traverse a cache C and a loss detector L. Both just forward packets that they receive, but they carry out additional tasks: the cache additionally stores a copy of them, and the loss detector confirms the successful reception of each packet with an ACK message to C (again, no NACKs). ACKs from L arrive at C after rtt_c. If they do not arrive for a certain packet within a timeout rto_c, they are retransmitted. Introducing a cache and a loss detector divides the path into three segments, and there is a certain loss probability on each of the path segments—p_1, p_2 and p_3.

This scenario very closely matches a recent proposal at the Internet Engineering Task Force (IETF) called LOOPS [7]. LOOPS, which stands for “Local Optimizations on Path Segments”, is a proposal to form a Working Group, whose first formal “Birds of a Feather” (BoF) meeting was held at the 105th IETF meeting in Montreal, Canada, in July 2019. The envisioned operation of a LOOPS system, described in [8], includes two modes. In tunnel mode, C in Figure 1 would encapsulate packets in addition to caching them, so that they can carry necessary LOOPS information such as sequence numbers (LOOPS promises not to examine the header of transport protocols, and therefore needs its own sequence number space). In transparent mode, C would simply forward packets and additionally cache them together with a hash identifier that is calculated from immutable header fields.

In both cases, L sends ACKs to C, and L may or may not re-sequence packets to ensure in-order delivery to the destination at the expense of latency. Additional considerations for LOOPS include the possible use of Forward Error Correction (FEC) on the path from C to L instead of plain retransmission, avoidance of congestion by limiting the rate at which C retransmits packets when it sees that the delay on the C-L path grows (ACKs from L include a timestamp for this purpose), and methods to inform the receiver R of congestion. Clearly, LOOPS systems could benefit from an understanding of the impact of the various factors shown in Figure 1—and some of them can be made available, e.g. to C via measurement. For example, the “spin bit” [9] offers a simple method for C to obtain rtt_s (which could be used to estimate rto_s) by monitoring packets.

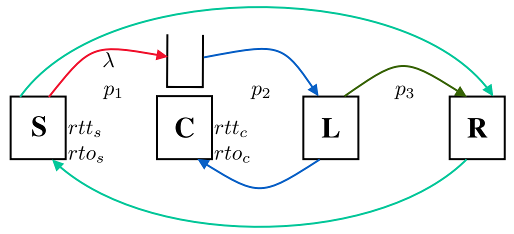


Fig. 1: LOOP architecture

We can see that the system that we investigate, depicted in Figure 1, is a subset of the possibilities that are envisioned for LOOPS: packets are not delayed for the sake of ordering, there is no FEC, just the simplest case of L ACKing and C retransmitting when it can. Thus, to indicate its relationship to LOOPS, yet clarify that it is more general and only covers a subset of LOOPS, we call our retransmission setup “**LOOP**”.

Recursive network architectures constitute a very different, but no less fitting use case. These architectures—the Recursive Internetworking Architecture (RINA) [10], the Recursive Network Architecture (RNA) [11] and the RINA-based compositional architecture described in [12]—all assume that all layers have to carry out the the same basic functions (albeit possibly instantiated in different ways, depending on the environment), and that layers can be organized at will, in a way that is suitable for a specific network scenario.

According to these architectures, the fact that both the link layer and the transport layer retransmit lost frames or packets is not an unnecessary duplication of functions, but it may instead be an integral element of communication, just carried out in slightly different ways at different scopes. In RINA, it is quite unusual to consider retransmissions (or any other function) only end-to-end; if C and L in Figure 1 would occur at a layer below S and R, they are senders and receivers in their own right, and it becomes natural to ask whether C should retransmit too. But then, more generally, what is the information that a sender should be given? E.g., for optimal operation, would C need to be told about p_1 or rto_s? If a sender is not at the topmost layer, how hard should it try to retransmit—for how long should it cache data?

Caching is a core element of Information-Centric Networking (ICN); it is therefore not surprising that proposals for local retransmissions exist in an ICN context too. However, also in an ICN context, the assurance of in-order delivery creates latency, which is a problem of mechanisms such as MFTP [13] or BELRP [14] that was addressed by the R^2T mechanism [15], making it the one that is closest to our model.

Irrespective of the use case, intuitively, the LOOP cache is just a queue which can be analysed using various queuing

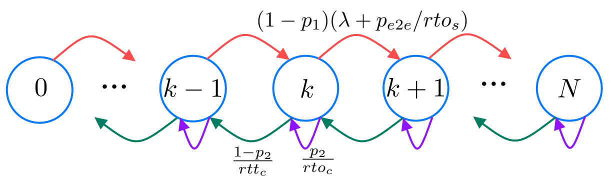


Fig. 2: CTMC for LOOP

TABLE I: Notations

Symbol	Description
λ	the packet send rate from S
μ	the service rate of C
μ_s	the service rate of S
ρ	the load at the buffer, i.e., $\frac{\lambda}{\mu}$
p_1	the probability of a packet to be dropped from S to C
p_2	the probability of a packet to be dropped from C to L
p_3	the probability of a packet to be dropped from L to R
p_b	the blocking probability, i.e. the probability of not caching the packet (the cache is full)
rtt_s	the Round-Trip Time (RTT) between S to R
rtt_c	the Round-Trip Time (RTT) between C to L
rto_s	the Retransmission Timeout (RTO) between S to R
rto_c	the Retransmission Timeout (RTO) between C to L
N	the maximum buffer size at C
k	the number of backlogged packets in the buffer at C
X	the random variable where $X = k$
T	the random variable for waiting time
R	the random variable for retransmission rate

models. Vishwanath et al. [16] used Markov chain analysis to model an $M/D/1/B$ queue for a network-on-chip output-queuing router. They proposed the method to predict the optimal buffer size using pre-defined design parameters. Barik et al. [17] used a stochastic model for deciding an optimal choice of the initial window of TCP flows depending on the bottleneck router's buffer size, number of active TCP connections, bottleneck router's link capacity, RTT, and flow size. Chandrayana et al. [18] proposed a mechanism to configure RED parameters by analyzing of an $M^X/M/1/B$ queuing model. Similarly, Dirnopoulos et al. [19] used a Markov model for TCP and $M^X/M/1/B$ model for a bottleneck router's buffer to investigate the burstiness of TCP.

III. LOOP Analysis

In this section, we present a Continuous-Time Markov Chain (CTMC) with finite states to model the system illustrated in Fig. 1. The notations we use throughout the paper are summarized in Table I. We assume that S transmits packets at a rate following a Poisson distribution. Despite its simplicity, it has been shown that congestion controllers such as TCP can be modeled with this distribution, especially when there are many long-lived TCP flows (usually in the core) competing with each other [16]. This is also the case in LOOP, which is placed in the network and many flows cross it. In addition, flows originated from congestion controllers such as LGC [20] that use packet pacing follow a Poisson process even from the sender. It has also been shown that even with short-scale bursty traffic, the use of fine-grained pacing timers can make TCP almost as smooth as a Poisson stream [21]. These results form our basis of utilizing Poisson and CTMC, and accordingly, we do not re-validate our model with simulations from this point of view. However, we focus on retransmissions, packet drops, and where they happen to analytically study the efficacy of LOOP.

The packet caching in LOOP can be modelled as an $M/D/1/N$ queuing model where the arrival is a Poisson process and the service is a deterministic process. Fig. 2 presents the Markov chain for a cache in LOOP where the chain has $N+1$ states, i.e., $0, 1, ..., N$. The newly unacknowledged packets arrive at C with the rate $(1-p_1)\lambda$ from S. Assuming the lost packets in between C to L to be recovered by LOOP, the end-to-end packet-loss probability measured by S is

$$p_{e2e} = p_1 + (1-p_1)p_b p_2 + (1-p_1)(1-p_b p_2)p_3 \ .$$

The rate of retransmitted packets (sent by S) that appears at C is $(1-p_1)\frac{p_{e2e}}{rto_s}$. Thus the total arrival rate of packets that grows the cache is given by

$$\lambda_1 = (1-p_1)(\lambda + \frac{p_{e2e}}{rto_s}) \ . \tag{1}$$

The incoming packets at C are cached and forwarded. Once these packets reach L, L acknowledges them, and when the acknowledgements arrive at C, C removes them from the cache. This occurs at a rate, i.e. the service rate of $\mu = \frac{1-p_2}{rtt_c}$. Retransmissions in LOOP occur in a deterministic time period of rto_c with probability p_2, which means that the rate is $\frac{p_2}{rto_c}$ at C.[2] Therefore, the load at C can be computed as $\rho = \frac{\lambda_1}{\mu}$. To obtain the steady state probabilities, this model follows similar to an $M/M/1/N$ queuing model (see [22] on how to obtain the solution), where the service rate is μ:

$$\pi_0 = \frac{1-\rho}{1-\rho^{N+1}} \ , \tag{2}$$

$$\pi_k = \rho^k \pi_0 \ , \quad k = 1, \cdots, N \tag{3}$$

$$E[X] = \frac{\rho}{1-\rho} - \frac{(N+1)\rho^{N+1}}{1-\rho^{N+1}} \ , \tag{4}$$

$$E[\lambda_1] = \lambda_1(1-\pi_N) \ , \tag{5}$$

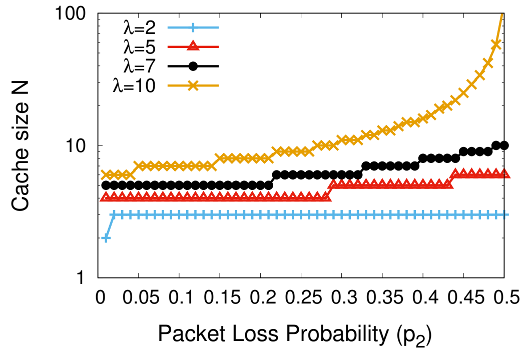
$$E[T] = \frac{E[X]}{E[\lambda_1]} \ , \tag{6}$$

$$p_b = \frac{(1-\rho)\rho^N}{1-\rho^{N+1}} \ , \tag{7}$$

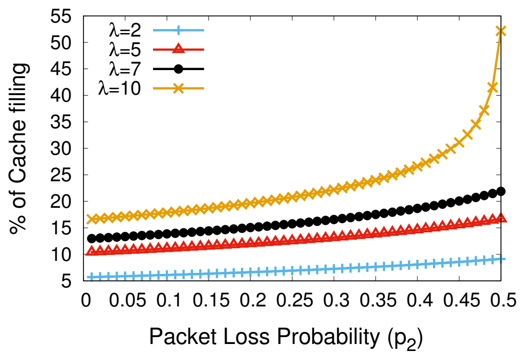
where π_k is the steady state probability of state k, $E[X]$ is the expected number of packets in the cache, $E[\lambda_1]$ is the expected arrival rate for caching (caching rate), and $E[T]$ is the expected waiting time of packets in the cache. Since p_b is already in p_{e2e}, fixed point iteration can be used with a given initial p_b to get the steady state blocking probability in (7).

We further divide our study into two parts depending on whether the cache size is set based on p_2 and p_b or fixed. A LOOP deployer can monitor the value of p_2, and depending on other parameters, (s)he can set a cache size for the LOOP, or a fixed cache size independent of p_2 and p_b can be deployed.

[2] Self-transitions are not usually drawn in CTMC since not exiting a state means staying in that state. However, we draw this special event to emphasize retransmissions from C, and removing them does not affect the model.



(a) The required cache size (N) at C (in packets)



(b) The filling percentage of the cache ($100\frac{E[X]}{N}\%$) as a function of p_2

Fig. 3: Evaluating LOOP with $p_1 = 0.01$, $p_3 = 0.01$, $rtt_s = 0.1$, $rto_s = 0.2$, $rtt_c = 0.05$, $rto_c = 0.1$, $p_b = 0.01$.

A. *Sizing the Cache as a Function of p_2 and p_b*

Here, we assume that for a given p_2, we need to guarantee a certain cache blocking probability. This actually answers the question how much buffer we need if with probability $1 - p_b$, packets should be successfully cached until they cross the second segment. Apparently, with probability p_b, a packet cannot be cached, so in case of loss, S has to retransmit it, not C. Solving (7) for a given p_b and p_2 yields

$$N = \left\lceil \log_\rho \frac{p_b}{1 - \rho + \rho p_b} \right\rceil \tag{8}$$

where $\lceil . \rceil$ denotes the ceiling function. We use fixed point iteration to get the steady state blocking probability p_b using (8) and (7). Throughout the study, we set the initial p_b to 0.01 (we iterate this to find its steady state value). Then for different values of p_2 and λ, we compute the cache size N as shown in Fig. 3(a). The figure indicates that with larger values of p_2, larger cache sizes will be required to maintain the desired blocking probability. A higher arrival rate also needs a larger cache size. The cache size difference is larger as p_2 grows.

Note that, in all of our performance evaluation, we investigate the full range of loss ratios from 0 to 1 only to obtain a better understanding of the system; this range is not necessarily realistic. In most real-life settings, the link between C and L will be a wired connection, where loss ratios in the order of 0.5 or above are not to be expected (although special use cases with high loss ratios may exist, e.g. in wireless sensor networks).

Fig. 3(b) plots the filling percentage of the cache, i.e. $100\frac{E[X]}{N}\%$. We use (8) to obtain N for different values of p_2 and λ. When p_2 increases, the network drops more packets and the cache later retransmits the lost packets which in turn increases the number of cached packets at C. A larger arrival rate also increases the number of cached packets. Again, we can observe that higher values of p_2 affect higher rates more, meaning that the difference becomes larger.

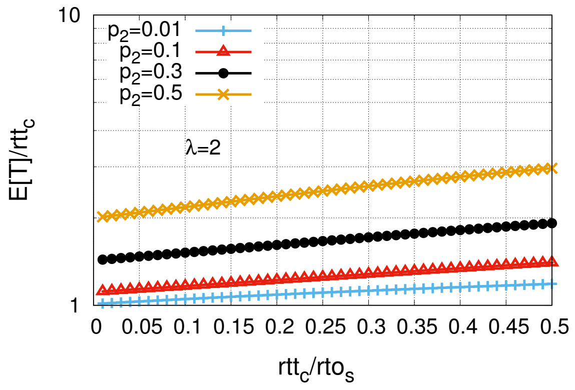
The RTT and RTO parameters that appear in the $M/D/1/N$ queuing model are rtt_c and rto_s. rtt_c influences the ACK rate, i.e. the service rate at C, whereas rto_s affects the retransmission rate from S. To evaluate how rtt_c affects the caching time, we vary rtt_c/rto_s in the range $[0.0, 0.5]$ and compute the expected waiting time of packets at C. Fig. 4 plots $\frac{E[T]}{rtt_c}$ (the expected waiting time, in units of local RTTs) for different values of rtt_c/rto_s, p_2, p_b, and λ, and we use (8) to obtain N. For a fixed value of p_2 and λ, a higher value of rtt_c increases the waiting time of packets. Similarly, larger values of p_2 and λ increase the expected waiting time. When $\lambda = 10$ and $p_2 = 0.5$, the plot is flattened after the ratio 0.25 because the cache is fully utilized and excess packets are not cached, the expected waiting time does not increase.

B. *Analysis of a Fixed Cache Size Independent of p_2 and p_b*

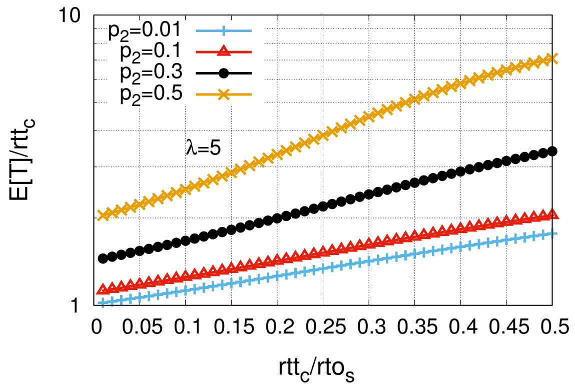
In this analysis, we do not obtain the cache size based on p_2 and p_b. Instead, we set N to some fixed values and then, we analyze the model for different values of λ, p_2, and rtt_c/rto_s.

Fig. 5 presents the filling percentage of the cache by varying p_2 for a set of N and λ. Obviously, p_b, calculated by (7), will be different for each value of N. A larger value of p_2 fills the cache faster than a smaller one. Higher arrival rates have larger impacts in filling the cache. From the figures, we observe that the cache is almost empty up to a certain value of p_2. However, small increments of p_2 can seriously affect the percentage after that. This implies that the increase is not linear, and one can expect almost full utilization with a slight change of p_2. The other finding from these figures is that the cache size does not play a significant role when p_2 exceeds a certain value because the arrival rate and also the service rate are fixed, and the cache, irrespective of its size, will become almost fully utilized after a short time. This explains why the plotted lines for different values of N are similar. Intuitively, the size of the cache can only play a role when loss happens intermittently (e.g., when loss increases to a certain value for a brief period). This may well happen in practice. Our model, however, considers the long-term behavior, where sustained loss is either too large or not too large to temporarily cache enough data, given a certain incoming rate.

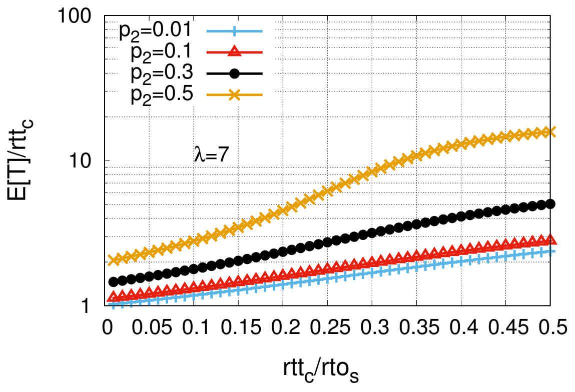
We then study the influence of rtt_c/rto_s on the waiting time in the cache. Fig. 6 plots $\frac{E[T]}{rtt_c}$ for packets in the cache. Fig. 6(a) shows that the maximum cache size has the least impact on the waiting time as all the lines overlap. A smaller value of rtt_c favors a smaller waiting time. Fig. 6(b) shows



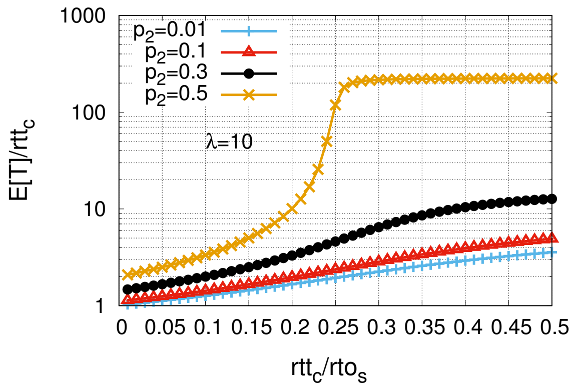
(a) $\lambda = 2$



(b) $\lambda = 5$



(c) $\lambda = 7$



(d) $\lambda = 10$

Fig. 4: The expected number of local RTTs that packets will have to wait at C, $E[T]/rtt_c$, with $p_1 = 0.01$, $p_3 = 0.01$, $rtt_s = 0.1$, $rto_s = 0.2$, $p_b = 0.01$.

that a smaller value of p_2 reduces the waiting time. We also observe that when $rtt_c/rto_s \leq 0.25$, for moderate p_2 values, the waiting time is less than twice of rtt_c.

IV. End-System Analysis

The end-system's buffer could be modelled as an $M/D/1/\infty$ queue where the arrival rate is the same as the packet send rate λ and the load is given by $\rho = \frac{\lambda}{\mu_s}$ where μ_s is the deterministic service rate. The CTMC of the queuing model is shown in Fig. 7 where the Markov chain has an infinite number of states. In the absence of LOOP in the network (i.e. S is solely responsible for retransmitting lost packets), the packet-loss probability between C and L, p_2, appears at the sender side. The service rate is the rate at which packets are removed from the buffer once they are acknowledged by R (the transition rate from state k to $k-1$). The service rate and the retransmission rate of lost packets (i.e. the transition rate from state k to k due to lost packets) are given by

$$\mu_s = \frac{(1-p_1)(1-p_2)(1-p_3)}{rtt_s} , \tag{9}$$

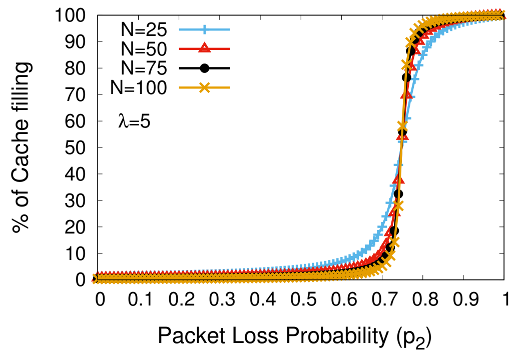
$$R = \frac{1-(1-p_1)(1-p_2)(1-p_3)}{rto_s} . \tag{10}$$

However, with LOOP in the network and the blocking probability p_b, we obtain μ_s and the expected retransmission rate R as

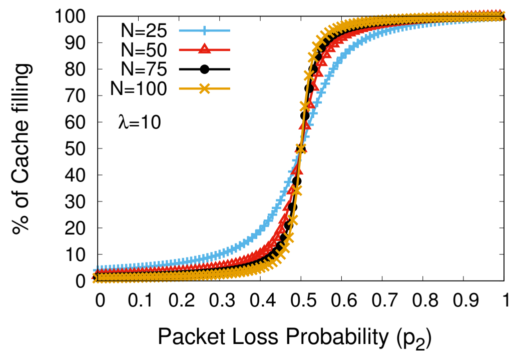
$$\mu_s = \frac{1-p_{e2e}}{rtt_s} , \tag{11}$$

$$R = \frac{p_{e2e}}{rto_s} . \tag{12}$$

For both the cases, finding the steady state probabilities follows similar to an $M/M/1/\infty$ queuing model. The difference between the two cases are: in networks with LOOP, S *sees* a packet loss probability of $p_2\, p_b$ between C and L whereas without LOOP, S gets a packet loss probability of p_2 from that segment. Now, the expected waiting time is $E[T] = \frac{1}{\mu_s - \lambda}$. Fig. 8(a) plots the expected waiting time of packets at the sender, i.e. the queuing time for different values of packet-loss probability p_2 with and without LOOP in the network. We set $\lambda = 5$ and p_2 varies up to 0.4; this assures the buffer at C is not saturated to be able to compare the two cases. Without LOOP, the waiting time at S increases with increasing p_2. Similarly the retransmission rate $E[R]$ also increases linearly with an



(a) $\lambda = 5$



(b) $\lambda = 10$

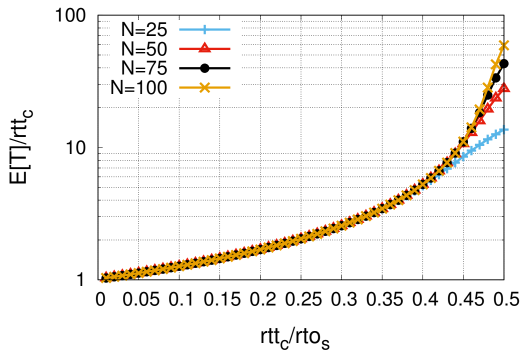
Fig. 5: The filling percentage of the cache as a function of p_2 with $p_1 = 0.01$, $p_3 = 0.01$, $rtt_s = 0.1$, $rto_s = 0.2$, $rtt_c = 0.05$, $rto_c = 0.1$.

increase in p_2 as shown in Fig. 8(b). However, the increase in these two metrics due to the increase in p_2 is negligible; this is the reason why the plots belonging to the LOOP analysis seem almost invariant in these figures, confirming the importance of such systems in the network compared to end-to-end systems.

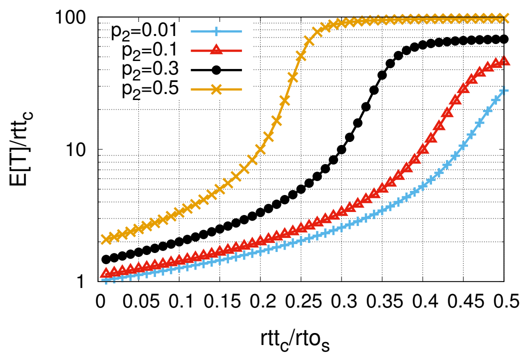
V. Conclusions

We have used Markov chains to evaluate a simple reliability improvement where a node along an end-to-end path caches packets and retransmits them when another node on the path does not acknowledge their reception within a timeout. The efficacy of such a mechanism depends on the packet loss probabilities of the path segment between the cache and loss detector as well as the ratio between the RTT of the inner and the RTO of the outer (end-to-end) control loop.

Using our model, we first obtained the required cache size depending on the packet loss probability in LOOP and cache blocking probability. We also observed the non-linear dependence between filling the cache and the packet loss probability, which means that a small increment in the packet loss probability above a certain value in LOOP can significantly affect the percentage of cache filling. The study also revealed that the configured cache size has the least long-term impact on the percentage of cache filling because irrespective of the



(a) Varying the cache size



(b) Varying p_2

Fig. 6: Plotting $E[T]/rtt_c$ with $\lambda = 10$, $p_1 = 0.01$, $p_3 = 0.01$, $rtt_s = 0.1$, $rto_s = 0.2$, and $p_b = 0.01$.

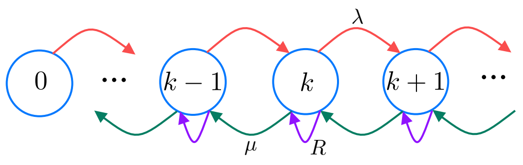
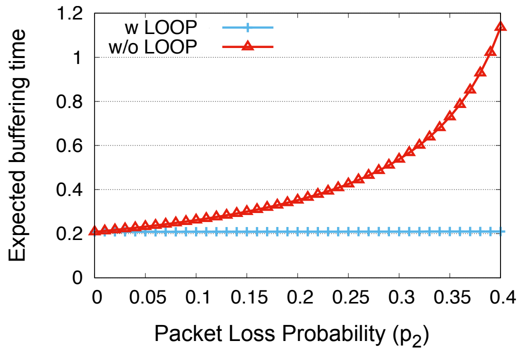


Fig. 7: CTMC $(M/D/1/\infty)$ at S

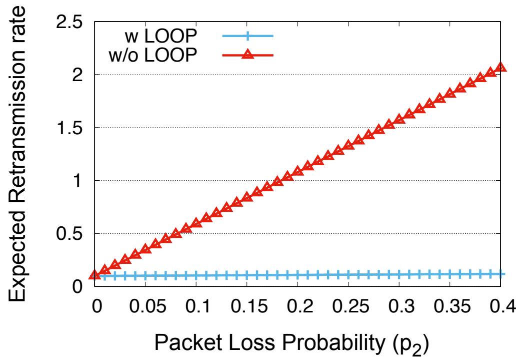
size, the cache can be fully utilized at higher packet loss probabilities.

We also studied the influence of the RTT of the LOOP segment on the packet caching time while the RTT is set as a function of the end-to-end RTO of the sender. A higher value of RTT increased the packet caching time while the configured cache size has the least impact on it. *With LOOP in the network, an end-system gets a smaller retransmission rate and expected caching time for packets than without LOOP being deployed in the network.*

As we have discussed in the outset, in-network retransmissions are often combined with a re-sequencing logic that ensures that packets maintain the correct order. This operation comes at the expense of added delay, and in a practical setting, it can be advantageous or not, depending on the situation. E.g., *modern* TCP implementations are robust against *some degree* of re-ordering. However, increased re-ordering robustness in TCP is not an across-the-board benefit either, as it can come



(a) Expected waiting time of packets at S



(b) Retransmission rate at S

Fig. 8: Evaluating the sender with $\lambda = 5$, $p_1 = 0.01$, $p_3 = 0.01$, $rtt_s = 0.1$, $rto_s = 0.2$, $p_b = 0.01$.

at the cost of an overall delayed congestion control response (depending on the specific mechanism). There is an interesting trade-off here: when packet loss is small and the inner loop's RTT is much smaller than the outer loop's RTT, the negative effect of re-sequencing can become tiny. Then, re-sequencing can possibly outweigh the disadvantages of, e.g., an older TCP implementation that might overreact to re-ordering.

In future work, we plan to extend our model such that it describes the trade-offs involved in such re-sequencing. Several other model extensions are also worth considering: for example, our model is limited to positive ACKs and timeouts—how will it change if we use NACKs that indicate holes in the sequence number space before a timeout would fire? Also, we have only considered a simple in-network retransmission loop. In reality, these loops could be used in sequence or even nested (such considerations have, in fact, already been brought forward in the IETF LOOPS proposal [7]).

Future work should also quantify the downsides of introducing LOOP: the higher network load caused by possible duplicate packets that it may bring about, and the overhead of the additional per-packet ACKs. Because of the additional load produced by LOOP, there will also need to be some form of congestion control between C and L, and a way to avoid "hiding" congestion from the end systems.

Acknowledgments

The authors were part-funded by the Research Council of Norway under its "Toppforsk" programme through the "OCARINA" project (http://www.mn.uio.no/ifi/english/research/projects/ocarina/). The views expressed are solely those of the authors.

References

[1] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end Arguments in System Design," *ACM Trans. Comput. Syst.*, vol. 2, no. 4, Nov. 1984.

[2] K.-C. Leung, C. Lai, V. O. K. Li, and D. Yang, "A packet-reordering solution to wireless losses in transmission control protocol," *Wireless Networks*, vol. 19, no. 7, pp. 1577–1593, Oct 2013.

[3] R. Ludwig and M. Meyer, "The Eifel Detection Algorithm for TCP," RFC 3522 (Experimental), RFC Editor, Fremont, CA, USA, Apr. 2003.

[4] R. Ludwig and A. Gurtov, "The Eifel Response Algorithm for TCP," RFC 4015 (Proposed Standard), pp. 1–13, Feb. 2005.

[5] P. Sarolahti, M. Kojo, K. Yamamoto, and M. Hata, "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP," RFC 5682 (Proposed Standard), Sep. 2009.

[6] Y. Cheng, N. Cardwell, N. Dukkipati, and P. Jha, "RACK: a time-based fast loss detection algorithm for TCP," IETF, Internet-Draft draft-ietf-tcpm-rack-05, Apr. 2019, work in Progress.

[7] L. Yizhou, X. Zhou, M. Boucadair, and J. Wang, "LOOPS (Localized Optimizations on Path Segments) Problem Statement and Opportunities for Network-Assisted Performance Enhancement," Internet Engineering Task Force, Internet-Draft draft-li-tsvwg-loops-problem-opportunities-03, Jul. 2019, work in Progress.

[8] M. Welzl and C. Bormann, "LOOPS Generic Information Set," Internet Engineering Task Force, Internet-Draft draft-welzl-loops-gen-info-01, Sep. 2019, work in Progress.

[9] P. De Vaere, T. Bühler, M. Kühlewind, and B. Trammell, "Three Bits Suffice: Explicit Support for Passive Measurement of Internet Latency in QUIC and TCP," in *Proc. IMC*. New York, NY, USA: ACM, 2018.

[10] J. Day, *Patterns in Network Architecture: A Return to Fundamentals*. Prentice Hall, 2007.

[11] J. Touch, Y. Wang, and V. Pingali, "A recursive network architecture," ISI, Tech. Rep. ISI-TR-626, 2006. [Online]. Available: https://www.strayalpha.com/pubs/isi-tr-626.pdf

[12] P. Zave and J. Rexford, "The Compositional Architecture of the Internet," *Commun. ACM*, vol. 62, no. 3, pp. 78–87, Feb. 2019.

[13] K. Su, F. Bronzino, K. K. Ramakrishnan, and D. Raychaudhuri, "MFTP: a clean-slate transport protocol for the information centric mobilityfirst network," in *Proc. ICN*. New York, NY, USA: ACM, 2015.

[14] S. Vusirikala, S. Mastorakis, A. Afanasyev, and L. Zhang, "Hop-by-hop best effort link layer reliability in Named Data Networking," NDN, Tech. Report NDN-0041, August 2016.

[15] Z. Wang, H. Luo, H. Zhou, and J. Li, "R2t: A rapid and reliable hop-by-hop transport mechanism for information-centric networking," *IEEE Access*, vol. 6, pp. 15 311–15 325, 2018.

[16] A. Vishwanath, V. Sivaraman, and G. N. Rouskas, "Considerations for sizing buffers in optical packet switched networks," in *IEEE INFOCOM 2009*, April 2009, pp. 1323–1331.

[17] R. Barik, D. M. Divakaran, and M. Welzl, "Characterizing the effects of tcp's initial window," in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, May 2017, pp. 886–891.

[18] K. Chandrayana, B. Sikdar, and S. Kalyanaraman, "Scalable configuration of RED queue parameters," in *High Performance Switching and Routing, 2001 IEEE Workshop on*, 2001, pp. 185 –189.

[19] P. Dirnopoulos, P. Zeephongsekul, and Z. Tari, "Modeling the burstiness of TCP," in *MASCOTS 2004*, oct. 2004, pp. 175 – 183.

[20] P. Teymoori, D. Hayes, M. Welzl, and S. Gjessing, "Even lower latency, even better fairness: Logistic growth congestion control in datacenters," in *IEEE LCN*. IEEE, 2016, pp. 10–18.

[21] H. Jiang and C. Dovrolis, "Why is the internet traffic bursty in short time scales?" in *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1. ACM, 2005, pp. 241–252.

[22] H. Elmiligi, M. W. El-Kharashi, and F. Gebali, "Modeling and implementation of an output-queuing router for networks-on-chips," in *Embedded Software and Systems*, Y.-H. Lee, H.-N. Kim, J. Kim, Y. Park, L. T. Yang, and S. W. Kim, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 241–248.