Transport Congestion Events Detection (TCED): Towards Decorrelating Congestion Detection from TCP

Pascal Anelli, Emmanuel Lochin, Fanilo Harivelo, Dino Martín Lopez Pacheco

Transport Congestion Events Detection (TCED): Towards Decorrelating Congestion Detection from TCP

Pascal Anelli
Université de la Réunion; LIM; France
pascal.anelli@univ-reunion.fr

Fanilo Harivelo
Université de la Réunion; LIM; France
fanilo.harivelo@univ-reunion.fr

Emmanuel Lochin
CNRS; LAAS; Université de Toulouse; ISAE; France
emmanuel.lochin@isae.fr

Dino Martin Lopez Pacheco
I3S; Université de Nice; France
dino.lopez@unice.fr

ABSTRACT

TCP(*Transmission Control Protocol*) uses a loss-based algorithm to estimate whether the network is congested or not. The main difficulty for this algorithm is to distinguish spurious from real network congestion events. Other research studies have proposed to enhance the reliability of this congestion estimation by modifying the internal TCP algorithm. In this paper, we propose an original congestion event algorithm implemented independently of the TCP source code. Basically, we propose a modular architecture to implement a congestion event detection algorithm to cope with the increasing complexity of the TCP code and we use it to understand why some spurious congestion events might not be detected in some complex cases. We show that our proposal is able to increase the reliability of TCP NewReno congestion detection algorithm that might help to the design of detection criterion independent of the TCP code. We find out that solutions based only on RTT (*Round-Trip Time*) estimation are not accurate enough to cover all existing cases. Furthermore, we evaluate our algorithm with and without network reordering where other inaccuracies, not previously identified, occur.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Transport Mechanisms

General Terms

Transport Protocol

Keywords

TCP, Congestion Event, Measurements

1. INTRODUCTION

TCP has the capability to adapt its sending throughput to the changing bandwidth available following the principle described in [11]. TCP considers a loss of a segment as a congestion in the network. A congestion event (or loss event) corresponds to one or several losses (or, in the context of ECN (*Explicit Congestion Notification*) [19]: at least one ACK (*acknowledgment)* packet with an ECN-echo) occurring in one TCP window during one current RTT period [6].

TCP congestion events play a role in terms of throughput performance as the congestion event involves a multiplicative decrease from the source. It is also important to clearly distinguish the loss ratio from the congestion ratio. Indeed, the number of losses during one RTT is not taken into account to characterize a congestion event as this number only impacts on the recovery time period.

TCP is strongly sensitive to spurious timeout [21] which trigger spurious retransmissions and result in a throughput decrease. Spurious timeout occurs when a non lost packet is retransmitted due to a sudden RTT increase (handover, route fluttering, network reordering, ...) which implies an expiration of the retransmission timer [18] set with a previous, and thus outdated, RTT value. This effect is known to be the root cause of spurious retransmission [20]. Several research work have raised this problem [15, 21, 3, 22].

In this paper, we propose a novel transport layer architecture where the congestion event detection algorithm is realised independently of the TCP code and detail the essential brick of this proposal: the congestion events detection mechanism. We aim to illustrate the feasibility of this concept by demonstrating that we can either obtain similar performances or also improve the accuracy of this detection outside the TCP stack. The main idea is to determine CE (i.e. the congestion detection) which impact on the TCP flow performance by monitoring the TCP flow itself. The principle is to obtain a detection system, at the edge of a network or at the sender-side which analyses the TCP behaviour through the observation of both data packets and

acknowledgments paths.

We implement this Implicit Congestion Notification (ICN) algorithm inside a framework that we call TCED (*Transport Congestion Events Detection*). Thus, TCED can be used either at the border of an autonomous system or conjointly within a TCP stack as a sublayer. ICN allows also to better understand and investigate the problem of congestion events estimation as well as proposing possible solutions to suppress inaccuracies of the current TCP loss-based algorithm. Following more exhaustive measurements, we show that an external congestion event detection is thus possible and present in section 2 the rationale of this design. Furthermore, we have identified that solution only based on RTT estimation are not accurate enough to cover all existing cases (*e.g* in particular retransmissions triggered at the begin of a connection). These results are provided in Section 4, just after the details of the algorithm (Section 3).

2. MOTIVATION TO BUILD A STAND-ALONE TCP CONGESTION EVENTS ALGORITHM

These last years, the number of TCP variants and minor extensions have greatly increase (see `[tcpm]`, `[iccrg]` and `[tsvwg]` IETF mailing lists). Some of them are deeply specialized to specific networks such as wireless LAN [16] or satellites links [5] while others focus on high speed networks [6, 23]. To date, except the historical and generic TCP Newreno/SACK (*Selective ACKnowledgment*) variant, there exists no universal TCP protocol able to perform indifferently over any kind of networks. The direct observable consequence of these many proposals is that **TCP source code is gaining in complexity** and that minor extensions proposed, such as for instance F-RTO (*Forward Retransmission TimeOut-recovery*) [21], do not help in terms of clarification of the source code. Furthermore, some improvements might be linked to a specific TCP version and cannot be deployed in the common TCP source code. Thus, the relevance of the OSI model is under question and in a recent paper [9], the authors argue that the transport layer should be now sliced in three sub-layers to cope with new network characteristics and the inherent complexity of the source code. All these reasons motivate the present study which aims at decorrelating the congestion events detection from the transport layer as presented in figure 1. In a sake of clear software engineering development, the main goal of this architecture is to simplify the task of kernel developers as well as improving TCP performances. Indeed, such architecture greatly facilitates protocol evolution and permits incremental rollout of congestion detection improvements. Finally, this scheme opens the door to another way to react to congestion by enabling ECN emulation at end-host. In this case, ICN emulates ECN marking to imply a congestion window reduction (see figure 1) with the same philosophy than in [4] where the authors enable AQM (*Active Queue Management*) emulation at end-host.

The target is to switch between the classical transport layer with the TCED layer architecture as illustrated figure 2.

3. IMPLICIT CONGESTION NOTIFICATION (ICN) ALGORITHM

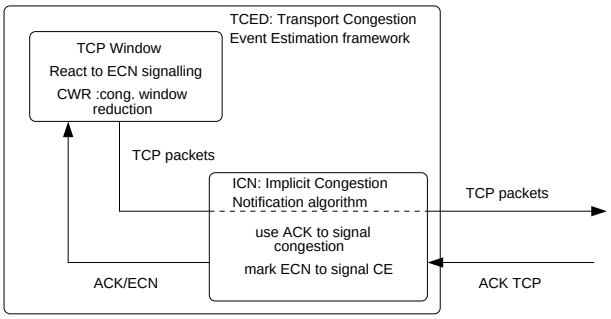

Figure 1: Decorrelating Congestion Detection from the Transport Layer.

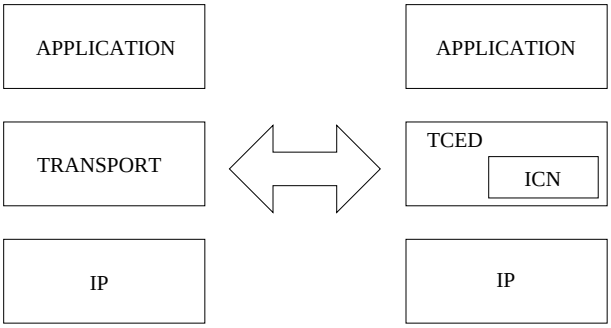

Figure 2: Re-architecture of the transport layer.

This section presents the design of our proposal and details the algorithm.

3.1 Design hypothesis

The design hypothesis is related to the localization of the Congestion Event (CE) estimator (i.e. from the source or from a node immediately connected to the source before the edge router). In this case, the estimator must be on a symmetrical path (ACK and data segments are received by the node) and the resulting RTT estimated is thus similar to the TCP connection. As our estimator operates to a live analysis (and not over past captured traces), only traffic at the sender side must be analyzed.

As TCP is a reliable transport protocol, lost data are obviously retransmitted. Thus, a CE can be identified from the TCP retransmitted packets. However, neither all of the retransmissions do not always indicate a loss, nor all losses do not always signal a CE:

- When the TCP retransmission timer expires, TCP triggers a *Go Back N* recovery procedure which could lead to retransmissions of packets effectively received (*i.e.* spurious retransmissions). Furthermore, TCP can consider losses following network packets reordering or following a significant increase of the RTT (*e.g.* in case of vertical handoff[1]). In this particular case, known as

[1]In a mobile context, a vertical handoff occurs when a mobile node is moving from a low delay network such as wireless

spurious timeout, the ACK get back too late to reset the retransmission time. These false losses identifications strongly impact on the TCP overall performances in terms of achieved throughput;

- Each loss does not have to be taken into account when identifying a CE. Indeed, when multiple losses occur in a single window, only the first loss is needed to identify a CE and other losses inside the same window must be ignored. To realize this, we need to be able to detect the first loss and the size of the sending window.

The ICN algorithm allows to determine which loss affects the TCP flow from the capture of TCP segments.

3.2 Interpreting CE

As previously explained in the introduction, a congestion event is defined as a set of losses occurring on a TCP window which involves a TCP congestion adaptation during an RTT. As a data window is emitted during an RTT, the estimation of the window size allows to identify data transmitted during one RTT given. When there is a retransmission, the bottom of the window is set on the retransmitted segment of data while the top of the window corresponds to the highest sequence number data sent. Then, all retransmitted packets between these two bounds are considered as belonging to the same CE. Indeed, a CE starts when a loss occurs and stops when the top of the data window during this congestion notification is acknowledged.

Detecting only retransmitted packets is not enough to identify a loss. Indeed, we have to be sure that a retransmission is not due to a TCP error. This case must be taken into account to avoid an overestimation of the CE over path where high RTT variations and reordering occur.

In [20] and [2], the authors classify as unnecessary (spurious) retransmission, those acknowledged in a delay lower than $\alpha * RTT_{min}$ (where RTT_{min} is the lowest RTT measurement and $\alpha = \frac{3}{4}$). If the next new ACK arrives in delay lower than $\alpha * RTT_{min}$ after the retransmission, then it means the ACK was already in transit when the retransmission occurred, and the timeout was spurious. This delay is a key value in the detection process. Indeed, when the delay is very small, this can lead to interpret unnecessary retransmissions as losses. On the contrary, when the delay is close to the current RTT, a new ACK could be received and the retransmission would be considered as unnecessary. In this configuration, the number of CE can be underestimated. As above, the principle to identify retransmission is based on a waiting delay T before validation. Obviously, this method introduces an additional delay which can be considered as a trade-off between the reliability and swiftness of TCP in terms of losses detection. In this context, it exists a delay between the network congestion and its detection by ICN. The CE detection is twofold. First, the identification and then, the classification of a retransmission as a lost.

In brief, to obtain an accurate CE estimation we need:

- to accurately estimate the RTT to size the delay to validate a loss;
- to take into account the TCP window size to distinguish the loss triggering a CE from the loss occurring during a CE;
- to identify spurious retransmissions that should not be identified as lost packets;
- to manage multiple data retransmissions. This might append during severe congestions. In this case, multiple CE occur; we denote such situation of re-congestion.

3.3 ICN algorithm

Starting from the observation of the data segments and the ACK, we identify each CE from each TCP connection with a state machine. This state machine (given in Figure 3) identifies the control congestion phase and classifies retransmissions as spurious or not. TCP congestion control reacts following binary notification feedbacks allowing to assess whether the network is congested or not. The state machine used enters in two states as a function of these notification feedbacks:

1. the *normal state* which characterizes a TCP connection without losses. Following Karn [12] algorithm, we can estimate the RTT in this state, knowing that for each emitted segment, the delay is computed until the corresponding ACK is received. As soon as this estimation is done, the process restarts for the next transmitted segment. This RTT estimation is used to size the validation delay (T); In the context where connections start over a severely congested network, the losses of segments prevent to realize an RTT estimation. Then, the initial value is set arbitrarily and is set to the same value than the retransmission timer: 3 seconds [18];
2. the *congestion state* which starts from the loss of the first window data segment. Each time ICN enters in this state, a CE is counted for the TCP connection.

Two others temporary states are added between these both states. These states aim at identifying spurious retransmissions with the help of the validation delay as previously explained. Finally, when the top of the window (denoted *recover*[2] in Figure 3) is acknowledged, ICN enters in the normal state.

In the case where it is supposed that a retransmission of a packet has been lost, and such a packet is retransmitted again, ICN enters into a *re-congestion* state. Once in the re-congestion state, if the validation delay T expires before *recover* is acknowledged, ICN comes to the *congestion* state and the CE counter is increased in one unit.

In the context of using ECN flag, congestion are not deduced only from losses. However, for this kind of connection we need to extend our algorithm to analyze this ECN signalization. Otherwise, ICN will under estimate the CE ratio

LAN to a high delay network such as UMTS/GPRS. Due to the sudden RTT increase, spurious retransmissions might occur.

[2]Equivalent to the *recover* variable of TCP NewReno

which affects each TCP connection. A recent study proposes to help congestion events detection using ECN marking [22]. However and to the best of our knowledge, the deployment of ECN inside the Internet remains marginal. Indeed, the authors in [17] show that ECN flag is used only by 2.1% hosts in 2004 and all current systems do not enable ECN by default[3]. As a result, we choose to study this particular case in a future work.

It is important to note that ICN does not manage error control which remains under the responsibility of TCP. If some improvements concerning the retransmission decision are introduced inside TCP, as ICN does not depend on the error control, our proposal remains valid. As a result, ICN is a generic algorithm which does not depend on the TCP version used.

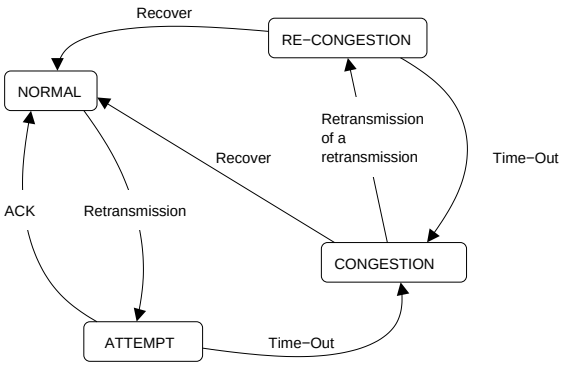

Figure 3: ICN state machine.

4. VALIDATION

We use the ns-2 simulator to estimate the notification error ratio as a function of the real CE. Our simulation model is able to apply different packet drop rates, different levels of statistical multiplexing and to introduce a level of packet reordering. The proposed scenario is motivated by the need to estimate ICN accuracy in the presence of spurious retransmissions and complex pattern packet drops. We aim at understanding the behaviour of ICN qualitatively rather than quantitatively. It means that we are not interested in determining the exact inaccuracy value through a statistical analysis but to determine a global accuracy trend.

The experiments are done over a simple dumbbell topology. A recent paper [13] provides a guideline to make a simulation model to evaluate TCP congestion control extensions. The model used afterwards follows these recommendations. We model the network traffic in terms of flows or sessions. Each flow corresponds to a HTTP request supported by a TCP connection. The *link load* is defined as follows:

$$\rho = \frac{\lambda E[\sigma]}{C}, \tag{1}$$

with C the bottleneck capacity. The traffic demand, expressed in bit rate, is the product of the flow rate arrival λ with the average flow size $E[\sigma]$. A reasonable fit to the heavy-tail distribution of the flow size observed in practice is provided by the Pareto distribution. The shape parameter is set to 1.3 for all simulations in the paper. As all flows are independent, the flow arrivals are modeled by a Poisson process. The load is changed by varying the arrival flow rate. Thus, the congestion level increases as a function of the load.

The load introduced in the network experiences different RTT (ranging from 59 to 250ms). To remove synchronization in TCP feedback and the phase effect, a traffic load of 10% is generated in the opposite direction. Measurements are saved after a "warm-up time" (*i.e* in the steady state) and the simulation duration corresponds to the reference flow duration. The bottleneck link capacity is set to 10$Mbps$. All others links have a capacity of 100$Mbps$. When the transient phase is finished, a long live flow of 400 packets is started, which represents the reference flow on which the ICN algorithm is applied. The minimum RTT experienced by the reference flow is around 100ms.

A packet reordering is applied on the reference flow such that randomly selected packets are randomly delayed. The network reordering is done after the bottleneck to maintain a same reorder rate for each network load. We adopt the delay distribution given in [24]. Delays are chosen from a normal distribution. The mean value is set to 50ms with a standard deviation of 16ms. Thus, the most chosen packets are delayed with values ranging from 0 to 100ms. The reorder rate used is 3.5%.

We make no claim about how realistic our background traffic is. We only want to reduce any simulation anomaly. The main objective of this model is to apply various and exhaustive congestion patterns to the analyzed flow. Figure 4 shows the conditions applied to the reference flow. The drop ratio results from packet drops of the background traffic without the reference flow. The reorder ratio is given from the measurements done on the reference flow.

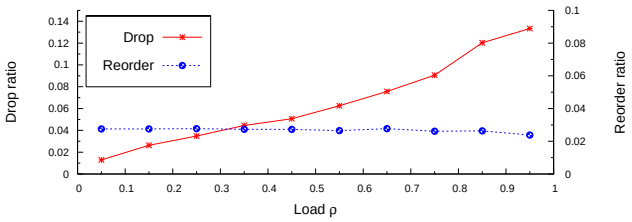

Figure 4: Scenario applied to the reference flow.

The accuracy of the CE detection is given by the inaccuracy parameter denoted ϵ and defined as follows:

$$\epsilon = \frac{N_{spu} + N_{und}}{N_{real} + N_{spu}} \text{ with } \epsilon \in [0, 1]. \tag{2}$$

With N_{spu}: the number of spurious CE identified either by ICN or TCP, N_{und}: the number of undetected real CE not identified by ICN or TCP and N_{real}: the number of effective (real) CE in the network. A real CE is deduced from the packets lost analysis (done *a posteriori*) occurring

[3]See Sally Floyd's ECN page for further details `http://www.icir.org/floyd/ecn.html`

in the network with the TCP window value when the lost is detected. When $\epsilon = 1$, it means that all detections are spurious and the more ϵ tends to 0, the more real congestions are detected.

We are not interested in determining the exact number of TCP losses as the LEAST algorithm presented in [1]. We aim at detecting when TCP congestion events occur in the network. In this work, we choose to compute a Loss Event Ratio (LER) (also called the congestion event rate [7]) defined as the ratio between the number of congestion event and the number of sent packets.

4.1 Choice of the validation delay

An important value which impacts on the ICN accuracy is the validation delay T. Three possible values are proposed to size T: RTT_{min}, $SRTT$ (the exponential mean of the RTT estimations) and RTT (the last RTT estimation). These values are adjusted according to a fraction called α as previously explained in Section 3.2). The goal is to find out the appropriate α that gives the best results in terms of CE identification.

We test the following α: $0.25, 0.5, 0.75, 1$. In the context of re-congestion, the validation delay will be tested with and without fraction. Fig. 5 gives the average inaccuracy over all possible combinations for a ρ background traffic ranging from 0.05 to 0.95 with a step of 0.1. All flows use TCP NewReno and the queue management for both routers is DropTail. When α is used, no validation delay is set. The inaccuracy in this case is the same as TCP. The α values are evaluated in growing order for each RTT base values. The dashed line in Fig. 5, represents the case where $\alpha = 1$ and when a re-congestion is detected.

As expected in our scenario, we can see that the network reorder introduces significantly errors in the congestion detection. The strong RTT variations produce spurious retransmissions. In this scenario, RTT_{min} is a good candidate to detect false CE notification. Near 0.25, ICN fails to detect spurious CE. Inversely when the validation delay is near RTT or $SRTT$, ICN fails to detect true CE. The distinction between congestion and re-congestion phases is not consistent. When the delay applied in the re-congestion state depends on $SRTT$ or RTT, the number of undetected CE increases. Congestion also decreases because ACK which correspond to a segment retransmitted are faster than the RTT measurement duration before the packet drop. The best trade-off for this scenario seems to choose an RTT_{min} validation delay. Indeed, this value leads to shorter detection delay while avoiding false retransmission identification at a low rate. In this experiment, normal congestion and re-congestion use the same validation delay. In the following, the validation delay is set to RTT_{min}.

4.2 Impact on different loads

Figure 6 shows the inaccuracy parameter as a function of load in the case where ICN uses RTT_{min}. To understand figure 6, we need to firstly look at figure 7. We define SLER, the TCP Spurious LER as the ratio between the number of spurious CE and the total number of sent packets. The SLER is almost constant in scenario with network reorder (because reorder rate does not change in our scenario). The

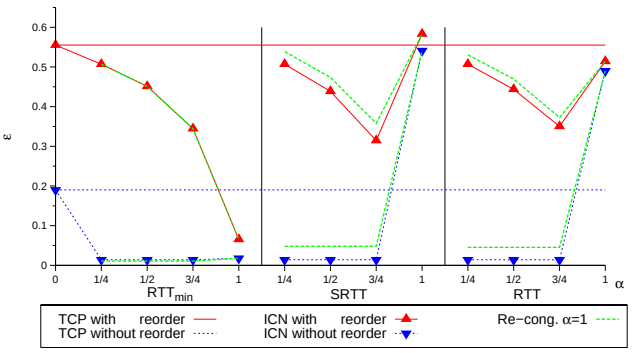

Figure 5: Sizing the validation delay.

LER increases with the load. Consequently and according to equation (2), the TCP inaccuracy decreases. For example, in case 0.05, no CE occurs and network reordering introduces spurious CE. This leads to an inaccuracy equals to 1.

In absence of network re-order, TCP inaccuracy is mainly due to spurious timeout in the Fast Recovery period. Indeed, with the impatient variant of TCP NewReno, the retransmission timer is only reset after the first partial ACK if a large number of packets are dropped from a data window [8]. The retransmission timer from the TCP sender will ultimately expire and the TCP sender will trigger a Slow-Start whatever the last retransmit packet outcome. The observed variations of inaccuracy in figure 6 come from unusual phenomenons as:

1. Real CE overlapping in a spurious CE. This case triggers a double error: one spurious CE and one undetected CE for TCP. This occurs when the bottom of the window is a re-ordered segment and inside this window a segment is lost. The retransmission of lost segment does not trigger a CE;
2. Packet reorder in the initial Slow-Start. This inserts spurious CE and let a congestion window smaller for TCP. For example, we can observe this for the 0.35 load, the accuracy seems better with reorder rather than without. In fact, ICN does not get the same number of errors but the number of CE in the scenario without network reorder is greater as shown figure 7. Surprisingly, reordering in Slow-Start avoids a burst of drops and a long delay to correct packet drops;
3. Spurious timeout from an ACK loss of a retransmission sent after a timer expiration. This case also triggers a spurious CE.

In a general manner, ICN is more accurate than TCP. Most of errors occur at the beginning of the flow when RTT_{min} are not correctly set. When a loss occurs in the first segments, there is a risk that a CE is not detected.

We have done measurements with TCP/SACK and reached similar conclusions. As SACK improves TCP retransmission decisions, there are less spurious retransmission which results in a better ICN accuracy.

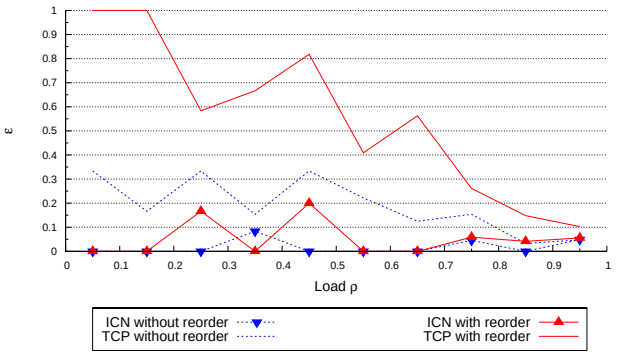

Figure 6: Inaccuracy function of load for TCP NewReno.

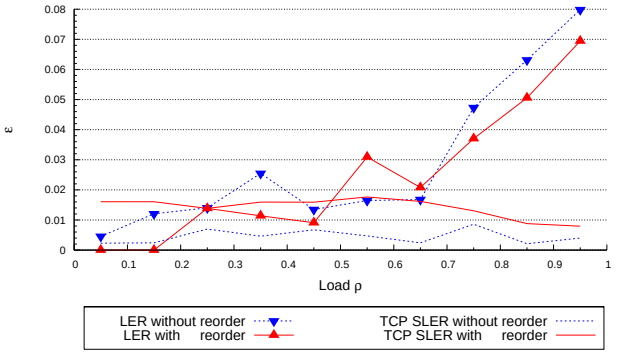

Figure 7: Reference flow of LER and TCP SLER function of the load.

4.3 ICN with timestamp

In order to distinguish an ACK resulting from a data packet to a retransmission packet, the Eifel algorithm [14] uses the timestamp option described in [10]. This option allows the sender to add timestamp to a packet later returned by the receiver in the corresponding ACK. Thus, the sender is able to compute an RTT by subtracting the current time to this timestamp.

The Eifel algorithm aims to distinguish an ACK which arrives after a retransmit has been sent in response to the original transmit or the retransmit. Although this timestamp scheme is not enabled by default mainly due to the overhead introduced by the option (i.e. 10 bytes in every packets) we choose to verify whether it might help the CE detection. As shown figure 8 and compare to 6, we can see this option improves the ICN detection but some identifications are still missing. For example, when the network reorder introduces a delay greater than twice the RTT (see load 0.45 in Figure 8), ICN cannot detect the spurious retransmission. As a result and whatever the algorithms used, a perfect detection is not realized today. The following section proposes to globally discuss all remaining cases.

4.4 Discussion

The off-line traces analysis shows that all congestions cannot be detected such as the case of spurious retransmis-

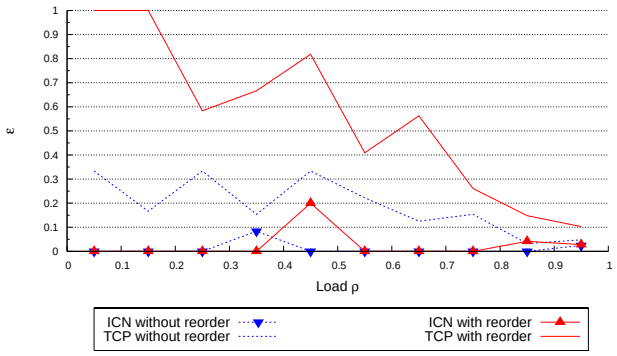

Figure 8: Use of the timestamp option.

sion dropped (which corresponds to the following sequence: transmit, retransmit, drop; denoted $(t-r-d)$). Indeed, the drop of a retransmission is already corrected by the initial transmission. From a transport protocol point of view, this sequence is similar to a spurious retransmission $(t-r)$. TCP has nothing to do (the first transmission is well-received). In the context of ICN, the $(t-r-d)$ sequence is an issue as ICN cannot detect a CE from this loss. This is an open problem since the congestion does not appear at the transport level but at the network level and thus, is out of the scope of the present study.

One important result is that a solution only based on the RTT is not really reliable at the beginning of the connection compared to ICN. Indeed, the RTT estimated is not accurate due to the weak number of possible measurements. This limits the use of such solutions to long-lived flows.

Globally and following the scenarios proposed, ICN algorithm gives good results. However, solutions based on a validation delay need an RTT higher or in the same order of magnitude than the reordering delay. If the difference between the RTT and the reordering delay is too large, it becomes difficult to detect spurious retransmissions. In this context, ICN algorithm reaches its limit of use. To get an accurate detection, the events' duration which causes spurious retransmissions must be less than an RTT.

5. CONCLUSION

This paper has proposed an algorithm (ICN) able to estimate congestion events occurring in the network and implemented as a stand-alone component inside a framework (TCED). The purpose of this scheme is to demonstrate that congestion event detection can be realized independently of the TCP code in a sake of better detecting congestion occuring in the network. This algorithm is based on the combined use of two realistics and feasible assumptions which are 1) a delay or a timestamp to validate a loss following retransmission and 2) the acknowledgments path. We have evaluated this algorithm with and without network reordering cases and shown that an external and live congestion events detection is possible. We also emphasized that previous solutions based only on RTT measurements are not able to cover all cases. In particular and following measurements done with the ICN algorithm, we show a lack of differenti-

ation between retransmissions due to reordering of loss and accuracy of the measurements with short TCP flows when using only RTT measurements.

Following, this work and the results obtained so far, we are currently planning the developpement of a kernel implementation of this framework and expect to drive a larger range of measurements which aims at benchmarking our proposal compared to embedded TCP spurious retransmission detection algorithms such as F-RTO and Eifel.

6. ACKNOWLEDGMENTS

We would like to thank Marc Allman from the ICSI Center for Internet Research for providing us the LEAST code. Pierre Ugo Tournoux for the development of the first ICN prototype and Tanguy Perennou for useful comments about this work.

7. ADDITIONAL AUTHORS

8. REFERENCES

[1] M. Allman, W. Eddy, and S. Ostermann. Estimating loss rates with tcp. *ACM SIGMETRICS Performance Evaluation Review*, 31(3):12–24, December 2003.

[2] Mark Allman and Vern Paxson. On estimating end-to-end network path properties. *Computer Communication Review*, 29(4), October 1999.

[3] S. Bhandarkar, A. L. N. Reddy, M. Allman, and E. Blanton. Improving the Robustness of TCP to Non-Congestion Events. RFC 4653 (Experimental), August 2006.

[4] Sumitha Bhandarkar, Narasimha Reddy, Yueping Zhang, and Dmitri Loguinov. Emulating aqm from end hosts. In *Proc. of ACM SIGCOMM*, 2007.

[5] Carlo Caini and Rosario Firrincieli. TCP hybla: a TCP enhancement for heterogeneous networks. *International Journal of Satellite Communications and Networking*, 22, 2004.

[6] S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649 (Experimental), December 2003.

[7] S. Floyd. Metrics for the Evaluation of Congestion Control Mechanisms. RFC 5166 (Informational), March 2008.

[8] S. Floyd, T. Henderson, and A. Gurtov. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 3782 (Proposed Standard), April 2004.

[9] Bryan Ford and Janardhan Iyengar. Breaking up the transport logjam. In *in Seventh ACM Workshop on Hot Topics in Networks (HotNets-VII)*, Calgary, Alberta, Canada, October 2008.

[10] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. RFC 1323 (Proposed Standard), May 1992.

[11] Van Jacobson. Congestion avoidance and control. In *Proc. of ACM SIGCOMM*, pages 314–329, Stanford, CA, August 1988.

[12] Phil Karn and Craig Partridge. Improving round-trip time estimates in reliable transport protocols. *ACM Computer Communications Review*, 17(5):2–7, 1987.

[13] Andrew Lachlan, Marcondes Cesar, and Floyd Sally. Towards a common tcp evaluation suite. In *PFLDnet*, 2008.

[14] R. Ludwig and M. Meyer. The Eifel Detection Algorithm for TCP. RFC 3522 (Experimental), April 2003.

[15] Reiner Ludwig and Randy H. Katz. The eifel algorithm: making TCP robust against spurious retransmissions. *SIGCOMM Comput. Commun. Rev.*, 30(1):30–36, 2000.

[16] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proc. of ACM MOBICOM*, 2001.

[17] A. Medina, M. Allman, and S. Floyd. Measuring the evolution of transport protocols in the internet. *Computer Communication Review*, 35(2), April 2005.

[18] V. Paxson and M. Allman. Computing TCP's Retransmission Timer. RFC 2988 (Proposed Standard), November 2000.

[19] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), September 2001.

[20] S. Rewaskar, J. Kaur, and F.D. Smith. A passive state-machine approach for accurate analysis of tcp out-of-sequence segments. *ACM Computer Communications Review*, 36(3):51–64, 2006.

[21] P. Sarolahti and M. Kojo. Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP and the Stream Control Transmission Protocol (SCTP). RFC 4138 (Experimental), August 2005.

[22] Michael Welzl. Using the ecn nonce to detect spurious loss events in TCP. In *Proc. of IEEE GLOBECOM*, December 2008.

[23] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *Proc. of IEEE INFOCOM*, 2004.

[24] M. Zhang, B. Karp, S. Floyd, and L. Peterson. RR-TCP: A reordering-robust TCP with DSACK. In *Proc. of the IEEE International Conference on Network Protocols - ICNP*, 2003.