Design and Validation of a Reliable Rate Based Transport Protocol: The Chameleon Protocol

Emmanuel Lochin, Guillaume Jourjon, Sebastien Ardon

Design and Validation of a Reliable Rate Based Transport Protocol: The Chameleon Protocol

Emmanuel Lochin
CNRS ; LAAS ;
7 avenue du colonel Roche, F-31077 Toulouse, France
[2] Université de Toulouse ;
UPS, INSA, INP, ISAE ; LAAS ;
F-31077 Toulouse, France
emmanuel.lochin@isae.fr

Guillaume Jourjon
NICTA,
Sydney, Australia,
guillaume.jourjon@nicta.com.au

Sébastien Ardon
NICTA
Sydney, Australia,
sebastien.ardon@nicta.com.au

***Abstract*—TFRC protocol has not been designed to enable reliability. Indeed, the birth of TFRC results from the need of a congestion controlled and realtime transport protocol in order to carry multimedia traffic. Historically, and following the anarchical deployment of congestion control mechanisms implemented on top of UDP protocol, the IETF decided to standardize such protocol in order to provide to multimedia applications developers a framework for their applications. In this paper, we propose to design a reliable rate-based transport protocol based on TFRC. This design is motivated by finding an alternative to TCP where its oscillating behaviour is known to be counterproductive over certain networks such as VANET. However, we found interesting results partly inherited from the smooth behaviour of TFRC in the context of wired networks. In particular, we show that TFRC can realize shorter data transfer compare to TCP over a complex and realistic topology. We firstly detail and fully benchmark our protocol in order to verify that our resulting prototype inherits from the good properties of TFRC in terms of TCP-friendliness. As a second contribution, we also propose a ns-2 implementation for testing purpose to the networking community. Following these preliminary tests, we drive a set of non-exhaustive experiments to illustrate some interesting behaviour of this protocol in the context of wired networks.**

I. Introduction

Transport protocols suffer of an evolutionary pressure due to the large diversity of recently identified networks properties. For instance, High Speed networks raise several performance problems while Delay Tolerant Networks are at the source of multiple new routing and end-to-end communications issues. As a result, the networking community shows a deep interest in rethinking the way to carry data over Internet. In particular, the IETF Transport Area Working Group (TSWG) is looking at specifying new congestion control and the recent Transport Architecture Evolution mailing list, discuss of ideas and issues surrounding the medium to long-term architectural evolution of the transport layer. Even the OSI model is now under question. Indeed, in a recent paper [1], the authors argue that the transport layer should be now sliced in three sub-layers to cope with new network characteristics.

Although Internet transport protocols must be redesigned due to congestion control lack of fairness and the difficulty of TCP to achieve high throughput, some issues must also be tackled in the context of satellite links, DTN and Intermittently connected Mobile Ad-hoc Networks (ICMAN). Those networks are notably characterised by long delay, lossy links and asymmetric path for which the classical transports protocol are known to poorly perform. New proposals such as Saratoga [2] and LTP-T [3] with pure ARQ scheme (indeed SACK or SNACK based), allowing a faster and efficient use of the available bandwidth in the case of one-hop transfer, have been proposed for interplanetary and satellites communications. However, we must remark that these protocols are deeply linked to the intrinsic store and forward nature of the DTN architecture and the characteristics of the link layer.

Furthermore, we can note that at least for the ICMAN scenario, a huge attention from the community have been given to the routing problem while giving a little interest to end to end transfer and transport. In dense PSN the radio channel might be lossy with variable bandwidth due concurrent communications and the fluctuating distance between nodes. A transfer protocol deployed in such an environment might take in account these parameters that lead to a problem sensibly different than the one solved by deep-space transfer protocol like Saratoga and LTP.

Recent work on transport protocols [4], [5] have proposed alternatives to the generally used window based congestion control. These protocols compute a sending rate which reproduces the TCP behaviour and have been defined as an alternative to UDP to carry multimedia traffic while respecting the fair-share principle introduced in [6]. In a wireless or lossy channels context, previous studies [7], [8] have demonstrated the poor TCP performances over wireless and multi-hop networks while others emphasise the good behaviour of rate controlled congestion control over these networks [9], [10]. Therefore, the design of a reliable rate-based transport protocol is perceived as a alternative data-delivery reliable service, that might be suitable for wireless multi-hop network such as vehicular networks (VANET) [11].

All these facts motivate the present study which detail a complete rate-based reliable transport protocol implementation as an alternative to the current domination of TCP in terms of reliable service.

There exist a range of reliability mechanisms from basic stop and wait to the more advanced Selective Acknowledgment mechanism [12], [13]. In a previous contribution [14], we outlined the design of a SACK-like mechanism, suited for rate-based congestion control such as TFRC [4]. While TFRC offers a smooth traffic dynamic property to the network, on the application side, this congestion control allows a direct exchange between the transport and application layers. Indeed, the metric used by both layer is identical (*i.e.* a sending rate in bit per second) and is not related anymore to the discrete and unusable value of window of packets per RTT. This previous study, mainly focused on the benefit of using such rate-based protocol to efficiently reach a negotiated throughput over a QoS-enable DiffServ network, did not specify the whole mechanism that could be used over a best-effort network. Thus, in this paper, we also consider the problem of flow control implementation, *i.e.* how to prevent packet loss at the receiver due to the receiving application not reading packet fast enough from the socket buffer. Such flow control mechanism is obviously mandatory to implement an efficient reliable transport protocol and require specific adaptations compare to the well-known TCP flow control version in order to be used conjointly with a rate-based congestion control mechanism. At last, the design of this implementation is not static and allows enabling or disabling the reliable mechanism plugged into TFRC. On the contrary, except in a recent study proposing a unreliable TCP mechanism (TCP-UREL [15]) to offer an alternative to DCCP/CCID#2 mechanism [16], none TCP versions allow to switch between a reliable and unreliable congestion control service.

The capability to directly interacts with the transport layer in order to either adapt the QoS required by applications and to enable or disable cross-layer mechanisms [17] and reliability have motivated the name of this protocol: Chameleon. Following the presentation of the whole structure of this new protocol in the following Section II we demonstrate that the present composition of SACK and TFRC does not impact on the TFRC TCP-friendliness property and validate the add-on of SACK and flow control mechanisms to TFRC by showing that there is no packet loss at the receiver, in case of a slow receiving application (Section IV). Before this, we discuss other possible designs in Section III. We investigate the behaviour of Chameleon protocol over a realistic topology which aims to represent an Internet provider backbone inspired from Free operator[1] in Section V. Finally, Section VI gives the conclusion and future work.

II. Specifications of the Chameleon protocol

We present in this section the protocol used for the integration of a flow control and, in particular, the composition of the TFRC congestion control and SACK reliable mechanisms.

The concept of Selective ACKnowledgments (SACK) was originally introduced in [12] as a TCP option that aims to optimize its reliable service by allowing a faster recovery in the case of a burst of lost packets [13]. By sending selective acknowledgements, the receiver of data can inform the sender which segments or packets have been successfully received and which ones have to be selectively retransmitted.

In [14], we have defined and validated the composition of SACK and TFRC mechanisms in order to provide a transport protocol compliant with the QoS negotiated with the network layer. This composition implied modification of the SACK mechanism to make it compatible to a datagram-oriented mechanism such as TFRC.

The modifications of the TFRC messages headers to integrate SACK are shown in Figure 1. The two first protocol data units represent respectively the TFRC header and the new header including SACK. The two last Protocol Data Units (PDUs) represent respectively the feedback for the classical TFRC protocol and for the TFRC-SACK composition. As TFRC mandates a new sequence number for each packet sent, we have to introduce a new identifier, linked to Application Data Units (ADUs), to perform the reliability. In the following and in the remaining of this paper, we refer to this identifier as sequence number. In all these headers, each field is either encoded over 4 or 8 bytes except for the `proto ID` (two bits), the `type` (two bits), `processing time` (one byte), and the `SACK payload` (variable length). We defined the datagram oriented SACK mechanism similarly to the stream oriented mechanism: the `SACK payload`, constituted by a sequence pair numbers. These pairs represent the edge of a continuous sequence of corrected received packets. The `length` represents the number of pairs in the `SACK payload`. Finally the `Offset` represents the sequence number of the first packet of the first pair.

In the following of the paper, we refer to TFRC with the SACK mechanism and the complete protocol with flow control as TFRC-FC-SACK, and TFRC with only the SACK mechanism as TFRC-SACK.

Based on this composition, the design of a flow control mechanism compatible with TFRC is presented and validated in the following.

Since the SACK mechanism requires receivers to maintain a buffer for the in-order delivery of packet to the application, we base our design on the introduction of a new window variable, *avail_win*, representing the space available in this buffer. This window should not be confused with the congestion-control window of TCP. The only purpose of this variable is to maintain, at the sender, the amount of buffer space available at the receiver and prevent the sender from transmitting more packets than there is available buffer space. Other candidate solutions for the design, including modification of the TFRC equation, are discussed in section III.

Figure 2 shows the sender and the receiver window. In this figure, the dark boxes represent data packets already sent or received.

At the sender, the flow control mechanism should stop transmitting data packets if the receiver's buffer is full. To achieve this, we use the *avail_win* variable, which, at the receiver, represents the available space in the receiver buffer,

[1] http://www.free.fr/

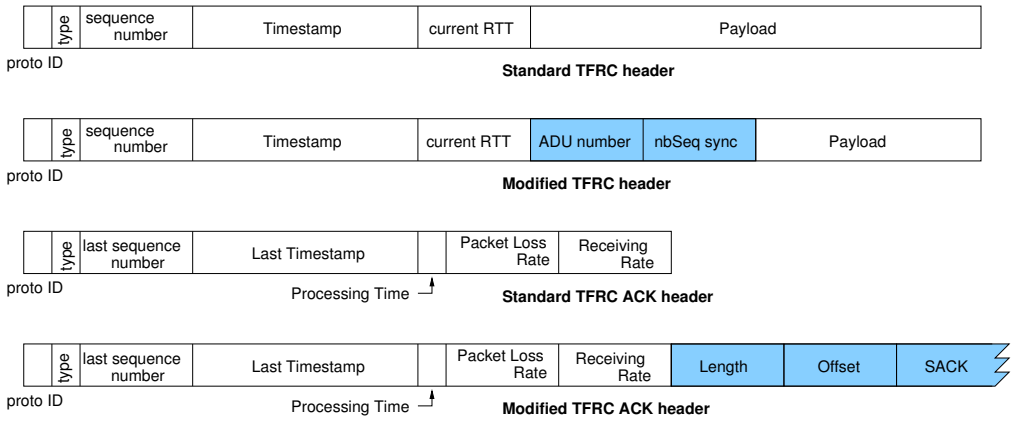
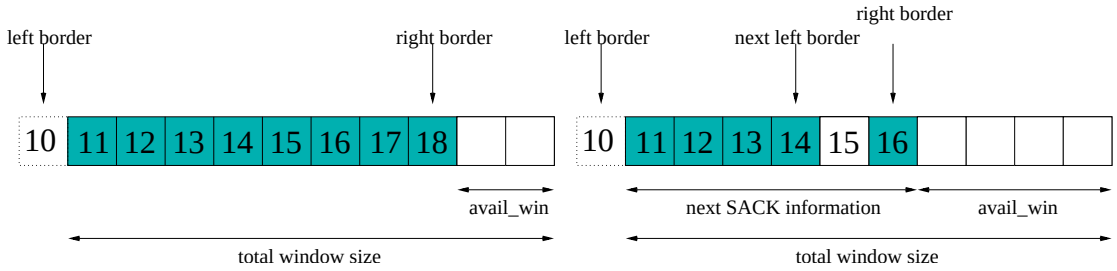


Figure 1. Modification in TFRC header



(a) The sender's window. *Left border:* highest acknowledged packet ID; *Right border:* highest packet ID sent so far. *avail_win:* available window size to send further data packets.

(b) The receiver's window. *Left border:* highest packet ID of the previously sent SACK vector; *Next left border:* highest packet ID of all correctly and in-order received packets. *Right border:* highest packet ID received so far. *avail_win:* available window size to receive further data packets.

Figure 2. The sender's and receiver's window

in number of packets. This variable is integrated in the TFRC-SACK feedback messages as a one-byte field after the `Receiving Rate` field of the last header in Figure 1. The *avail_win* variable therefore indicates, at the sender, the supposed number of packets which can be sent. *Avail_win* is never negative and upper bounded by the total window size. When this variable is non nil, the sender sends data packets at the rate computed by TFRC algorithm. Each time a packet is sent, *avail_win* is decreased by one at the sender. When *avail_win* is nil, the sender has already sent the maximum number of data packets which could have been accepted by the receiver. Note that the TFRC rate still condition the speed at which packets are sent, the *avail_win* variable condition the maximum number of packets which can be sent between receiving two feedback messages.

Indeed, as mentioned previously, each feedback message sent by the receiver contains the available buffer space. At the sender, upon reception of a feedback message, the *avail_win* variable is computed by withdrawing the number of packets sent since the header's `Offset` from the header's `avail_win`. A feedback message can therefore unfreeze the sender if the newly computed *avail_win* is non-nil or the SACK vector indicates that some packets need to be retransmitted.

At the receiver side, when a data packet is received, if its sequence number (S_{new}) is higher than the highest previously received sequence number (S_{old}), *avail_win* is reduced by $S_{new} - S_{old}$. Otherwise, this packet is out-of-order and is therefore placed in the reception buffer. When the application reads packets from the buffer, the *avail_win* is increased by the corresponding number of read packets.

III. Discussions

In this section, we discuss the design of our flow control for TFRC and explore alternative solutions.

The main feature of a rate-based congestion control mechanism is the use of an equation to determine the sending rate. This equation typically uses network measurements (or estimations) to calculate the theoretical rate at which TCP would send in similar conditions. Following this observation, we first investigated two other possible solutions to the flow control problem.

The first solution is to obtain the reading rate of the receiving application and send it back to the sender. This can be done either by estimating the reading behaviour of the application or by assuming that the application can communicate this reading rate to the transport protocol. The sender would then adjust its sending rate to the minimum between its computed congestion

control sending rate, twice the receiver's receiving rate, and the application's reading rate. However, this solution has two major drawbacks. Firstly, the reading rate depends on different parameters such as application type, CPU usage, etc. and may therefore follow complex patterns, which can be difficult to estimate. This may result in erroneous values leading in buffer overflow. Secondly, in order to provide packet ordering, the receiver temporarily buffers out of order packets. This can lead to a situation where the application's reading rate is nil, therefore the sender would stop even if there were space in the buffer.

The second possible solution would have been to modify the equation used to compute the sending rate. TFRC computes its rate (X) with an equation, defined in [4]. This equation takes into account the RTT and loss event rate (p). We note this equation:

$$X = F(p, RTT) \tag{1}$$

We propose to model the flow control impact on the transmitting rate as follows:

$$X = F(p, RTT) + G(avail\_win, p, RTT) \tag{2}$$

where $G(avail\_win, p, RTT)$ is a model of the flow control impact on the transmitting rate which takes into account the *avail_win* variable and $F(p, RTT)$ is the TFRC equation as defined in [4]. We chose not to pursue this solution, as it seems to introduce too much complexity to the rate computation. Moreover, if we want to avoid losing even one packet at the receiver due to a slow reading application, we would need to underestimate the sending rate, which would negatively impact on the protocol performance.

From this, it follows that there is no improvement or benefit in including the flow control in the TFRC's sending rate algorithm.

IV. Mechanism validation

In this section, we validate our Flow Control mechanism using simulation in ns-2.30. We first implemented the SACK-like mechanism within ns-2.30's TFRC. We also extended the ns-2 simulator to include the application layer to simulate an application reading from the socket buffer at different rate. Using this implementation, we conduct a set of simulations to demonstrate the effectiveness of our flow control mechanism, and quantify the potential impact of the SACK and flow control modifications over the TFRC flow dynamics.

A. TCP-friendliness conservation and Reliability

The first experiment aims at verifying the TCP-friendliness of TFRC-FC-SACK when sharing a bottleneck with other TCP flows. These days, the definition of the TCP-friendliness is still being debated [18]. In this study, we will first follow the definition in RFC3448: *"[...] a flow is "reasonably fair" if its sending rate is generally within a factor of two of the sending rate of a TCP flow under the same conditions."*. This definition concerns instantaneous values. Another common view is that, on average, a flow is TCP friendly if *the non-TCP source obtains a long-run term average sending rate not larger than the one TCP would have obtained under the same circumstances* [19].

To quantify the TCP-friendliness we therefore use an expression of the means ratio as shown on equation (3):

$$T(X) = \frac{\frac{1}{n}\sum_{i=1}^{n}\overline{x_i}}{\frac{1}{m}\sum_{i=1}^{m}\overline{y_i}} \tag{3}$$

where X is the protocol being studied, $\overline{x_i}$ the average throughput of the i^{th} X flow, n the number of X flows, $\overline{y_i}$ the average throughput of the i^{th} TCP flow and m the number of TCP flows. In this formula if T is inferior to 1 then the non-TCP flow is TCP-friendly, if T is equal to 1 then we have an ideal friendliness and finally if T if greater than 1 then the non-TCP flow overruns TCP.

In this simulation scenario, we use the butterfly topology shown on Figure 3. There are two sources transmitting to two destinations over a shared link between two intermediate nodes. These two flows are competing for the bottleneck link bandwidth. We perform two experiments where TFRC-FC-SACK is first competing with TCP-SACK, then with TCP New Reno. All three protocols are set to the same packet size of $1KByte$ and a maximum window size $64KBytes$. In both experiments, the application reading rate is infinite.

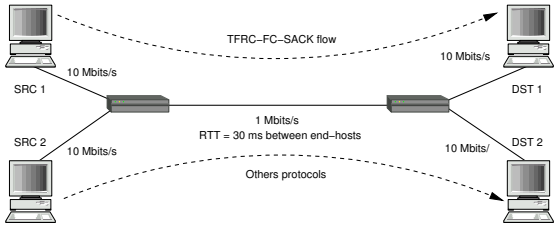
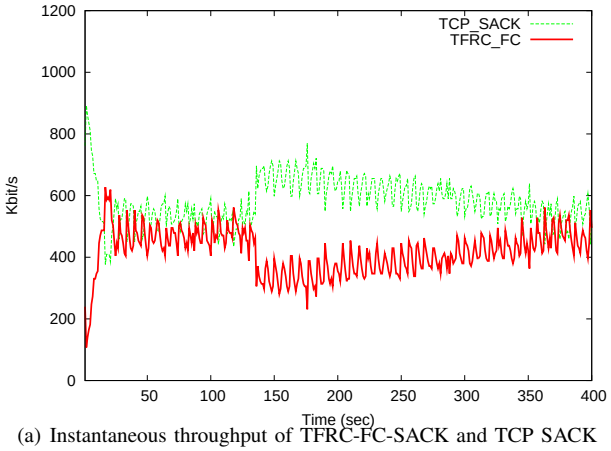


Figure 3. Topology of the scenario

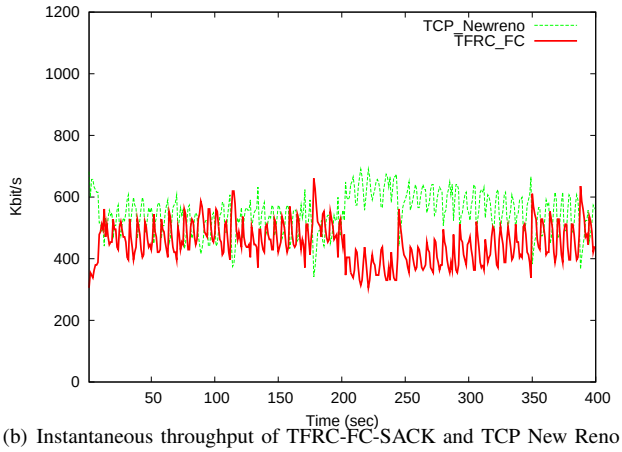
Results are presented in Figure 4. Each graph shows the flow instantaneous throughput at the receiver computed with an average sliding window throughput estimation with a $1ms$ window.

From Figure 4, we can see that the TFRC-FC-SACK flow instantaneous throughput is slightly inferior to both TCP SACK and TCP New Reno flows. In addition, the TFRC-FC-SACK instantaneous throughput is well within the $2x$ factor imposed by our TCP friendly definition previously mentioned. We can therefore conclude that this TFRC-FC-SACK implementation remains friendly with both TCP SACK and TCP New Reno[2]. In Figure 4 (a), TFRC-FC-SACK equally shares the bottleneck with TCP during almost $130s$. At $t = 130s$, TFRC suffers from consecutive losses and therefore sharply decreases its throughput. TFRC-FC-SACK then attempts to re-adjust its throughput to the equilibrium with TCP, but this process converges slowly, which is a well-known shortcoming of TFRC [20]. When competing against TCP New Reno, as shown in Figure 4 (b), TFRC-FC-SACK behaves similarly to

[2]Several experiments with different RTT and bottleneck capacity have confirmed these results. Due to space reason, we give in this section a general sample of these experiments.



(a) Instantaneous throughput of TFRC-FC-SACK and TCP SACK



(b) Instantaneous throughput of TFRC-FC-SACK and TCP New Reno

Figure 4. Validation of TFRC-FC-SACK composition in ns-2.30

its behaviour with TCP SACK except that it stays longer at the first equilibrium ($200s$ instead of $130s$). Furthermore after the consecutive losses TFRC-FC-SACK reaches the equilibrium with TCP New Reno faster than with TCP SACK. These differences can be explained as the TFRC equation models TCP Reno.

Table I presents the TCP-friendliness index of TFRC-FC-SACK calculated using equation (3). As all figures are below one. This confirms that TFRC-SACK is friendly with both version of TCP.

Table I
TCP-FRIENDLINESS INDEX RESULTS

TCP version	T(TFRC-SACK)
TCP/Newreno	0.82
TCP/SACK	0.72

For these experiments, we also validate the SACK mechanism, i.e. verify that all lost packets are retransmitted. In Table II, we summarize the number of sent and lost packets for each flow in the previous experiments. We can see from this table that TFRC-FC-SACK flows send less packets than both TCP versions. This is explained as the TCP flows overall throughputs are higher than the TFRC-SACK and the packet statistics are collected during a fixed time period of $400s$. Furthermore, we can see that the TFRC-FC-SACK flows experience less packets loss than both TCP flows (in absolute value and in percentage). This is explained by the fact that the rate-based congestion control mechanism produces a smoother sending rate compared to a window-based mechanism which is more aggressive. Finally, by using packet marking (not shown in the table), we verify that TFRC-FC-SACK retransmit all dropped packets until correctly received.

Table II
PACKETS STATISTICS

	number of sent packets	number of lost packet (percentage)
TCP/Newreno	26702	166 (0.62%)
TFRC-FC-SACK	21962	45 (0.2%)
TCP/SACK	28740	162 (0.55%)
TFRC-FC-SACK	20368	42 (0.2%)

B. Impact of the Application Read Rate

The objective of this experiment is to validate the flow control mechanism, by measuring the sender throughput when varying the application read rate, i.e. simulating a slower application. We also want to confirm that there is no packet lost due to a slow receiver unable to accept incoming packets. In addition, in this section, we quantify the impact of our SACK and Flow Control mechanisms over TFRC smoothness, by measuring the throughput stability during the data transfer.

In order to quantify this stability, we consider the average throughput for each time unit interval. For each time interval we compute each flow's throughput standard deviation [21] and obtain the following metric equation (4):

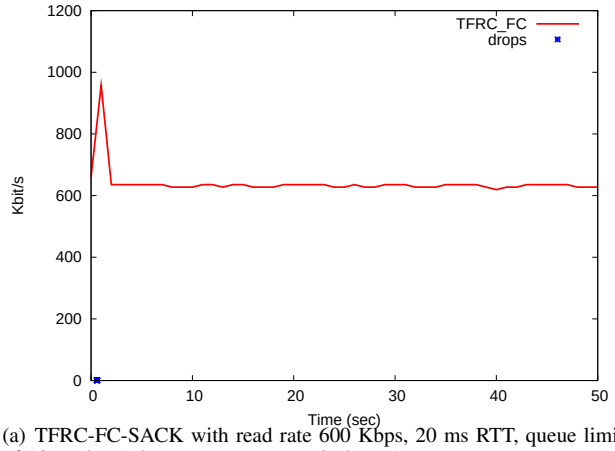
$$S = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{\overline{x_i}}\sqrt{\frac{1}{m-1}\sum_{j=1}^{m}(x_i(k)-\overline{x_i})^2}\right) \quad (4)$$

where $\overline{x_i}$ is the average throughput of the i^{th} TFRC-FC-SACK (resp. TFRC) flow, n is the number of flows, $x_i(k)$ is the throughput of the i^{th} TFRC-FC-SACK (resp. TFRC) flow for the k^{th} time interval and m is the number of time intervals.

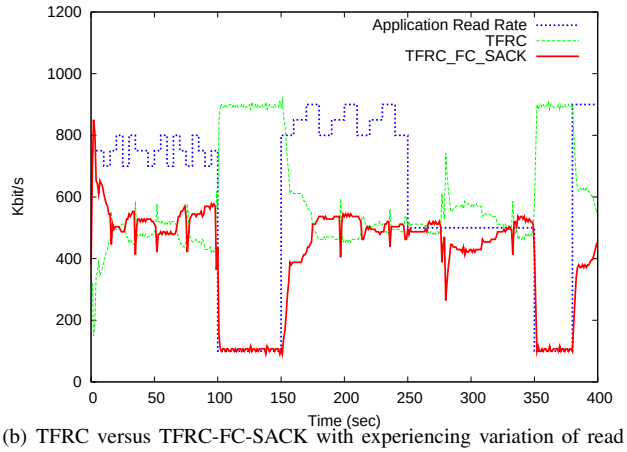
For these experiments, we use a simple topology where two nodes communicate through a third one. Packets are crossing two consecutive links of respectively 10 Mbps and 1Mbps bandwidth, for an overall 20 ms RTT (5ms delay on each link).

Figure 5 (a) shows the throughput of a TFRC-FC-SACK flow as the application read rate is set to $600kbit/s$ at the receiver.

Each packet loss event is illustrated on Figure 5 (a) by a cross on the x-axis. At the beginning of the transmission, the sender sends packets according to the slow start algorithm. This phase stops when the first packet loss event occurs. TFRC then enters the congestion avoidance phase. As soon as the receiver's buffer is full due to the application limited read



(a) TFRC-FC-SACK with read rate 600 Kbps, 20 ms RTT, queue limit of 10 packets (the cross represents six losses)



(b) TFRC versus TFRC-FC-SACK with experiencing variation of read rate

Figure 5. Validation of flow control mechanism

rate, the sender can no longer send further packets. As the application reads from the buffer non nil *avail_win* values are sent to the sender.

Hence, the sender is only allowed to send new packets when the receiver has delivered some packets to the application. Consequently, Figure 5 (a) confirms that the flow control mechanism operates correctly as the throughput is adapted to the receiver application read rate. Furthermore, Figure 5 (a) shows that the receiver does not drop any packets.

In Figure 5 (b), we mix one TFRC-FC-SACK and one TFRC flow in the same network conditions as previously. However, contrary to the previous experiment, the application read rate varies in time and follows a specific pattern as shown in Figure 5 (b). We have chosen this specific pattern as it represents a mix of above, below and equal to the fair share throughput.

From Figure 5 (b), we can first see that a read rate above to the theoretical fair share value ($500kbit/s$) does not impact on the behaviour of TFRC-FC-SACK: TFRC and TFRC-FC-SACK equally share the link bandwidth. Furthermore, the transition from this read rate to another one inferior to $500kbit/s$ does not induce any packet loss at the receiver buffer. Between $t = 100s$ and $t = 150s$, the application read rate is set to $100kbit/s$, i.e. under the theoretical fair share value. During this phase, we can see from Figure 5 (b) that TFRC-FC-SACK sending rate is following the application reading rate while TFRC flow can uses the rest of the bottleneck. At $t = 150s$, the application reading rate is set again to values above to the fair share for $100s$. We can see from the graph that during this period TFRC-FC-SACK and TFRC equally share the bottleneck bandwidth as expected. Finally, for the remaining variations of application reading rate, TFRC-FC-SACK continues to behave in a fair manner.

To quantify the impact flow control when the receiver application drives the transmission over the throughput smoothness, we use the stability metric, as defined in equation 4. We applied this criterion on a set of experiments that aims at checking that the flow control does not introduce any degradation in the smoothness characteristic of TFRC.

In Table III, we present the results of experiment when two identical flows share a bottleneck of $1Mbit/s$ during $400s$. We show in this table that TFRC-FC-Sack remains as smooth as TFRC when it is not limited by the application read rate. Furthermore when we introduce for both flows a read rate of $300Kbit/s$, the resulting stability of the system is increased. This result can be explained by the fact that the oscillations in the throughput are usually due to the congestion control mechanisms that tries to increase until the detection of a loss. In the case of a system limited by the application read rate the two flows do not try to increase nor decrease and therefore are more stable.

Table III
STABILITY INDEX FOR DIFFERENT PROTOCOLS

	TFRC	TFRC-FC-SACK	TFRC-FC-SACK read rate
S	0.094	0.097	0.051

V. EXPERIMENTATIONS OVER A REALISTIC TOPOLOGY

In order to evaluate our proposal over a more realistic case, a more complex topology is used in the next simulations. This topology is similar from this presented in [22]. According to a small study of the xDSL backbone of a Internet provider[3], most of these networks are built around a central core where several loops are connected. These loops are composed of a small number of routers. The aim of the closed loop is to have a fault tolerance.

For the simulation, a flower with five loops is considered as backbone, each loop has 8 routers, shown in figure 6. Each router (except the 5 core routers) has 2 DSLAMs connected to it, and each DSLAM has 3 hosts connected to it. Each link has the following characteristics: 10Mb/s bandwidth, 10ms delay, DropTail.

We emit 250 FTP over TCP/Newreno/Sack connections (flows) with random and non-identical hosts as source and destination. Then, we realise exactly the same experiments

[3]http://www.journaldufreenaute.fr/ftp/reseau_free_juin_2005.pdf

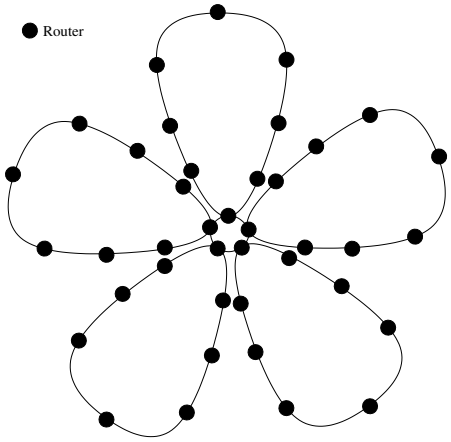


Figure 6. Backbone of the flower network.

	TCP-Newreno-Sack	TFRC-FC-Sack
Sum of transmission times (in seconds)	960.44	842.43
Time of the last last packet received (in seconds)	25.25	23.99

Table IV
GLOBAL METRICS

with TFRC/FC/Sack proposal. Each connection starts at a random time between 0 and 20 seconds and sends a random number between 10 and 600 packets.

Table IV provides the global result obtained for the whole simulation. We observe on average the transfer time of TFRC/FC/Sack flows is globally lower than TCP/Newreno/Sack.

In Figure 7, we classify each flow in ascending order of their size (*i.e.* the number of packets of each flow is ranging from 10 to 600 and is reported on the x-axis) and we report their respective time values on the y-axis. If several flows gets the same number of packets (as this number is randomly choosen), we report the highest value obtained by one of these flows. As we used the same seed for each simulation, the distribution of the flows' size was the same for both TCP/Newreno/Sack and TFRC/FC/Sack experiments. This figure tends to show that the transfer times of long TFRC/FC/Sack flows are smaller than TCP/Newreno/Sack flows. In order to better see this trend, in Figure 8, we classify in ascending order each transfer time value of each flow. Following these two figures, we clearly see that long TFRC/FC/Sack flows obtain a lower transfer time than TCP/Newreno/Sack. However, the shorter the TFRC/FC/Sack flows are, the higher their transit time. This means that TFRC/FC/Sack is not a good trade-off in the context of short data transfers such as HTTP requests but outperforms TFRC/FC/Sack in the context of long data transfers.

We explain this effect by the slow dynamic convergence of TFRC [20]. Indeed, TFRC is known to converge slowly to the

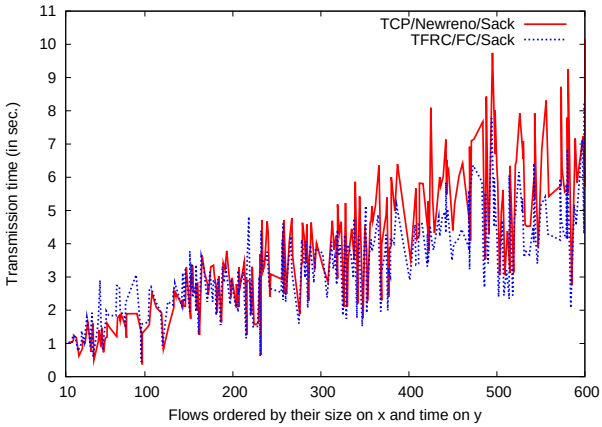


Figure 7. Transmission time of all the flows; as a function of the packet flow size (from 10 to 600 packets).

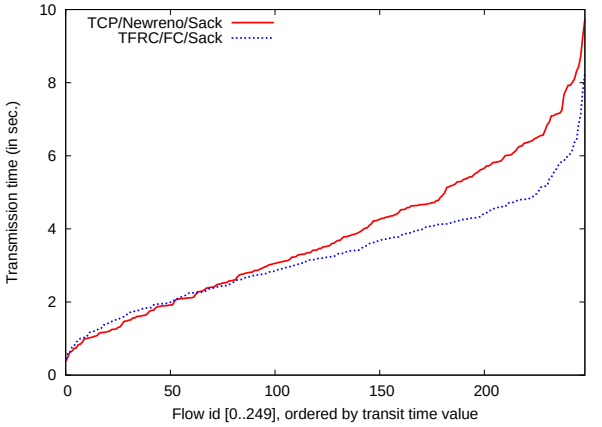


Figure 8. Transmission time obtained by all flows.

nominal capacity of the link over high delay and congested links just after the SlowStart phase. However, when TFRC enters the steady state, its long run behaviour is more stable than TCP which operates in a more oscillatory behaviour and then, might reduce several time its congestion window due to the highly dynamic character of the network.

In complement to the arguments presented in the introduction (Section I), these results show another interesting property of this protocol that might be helpful in high dynamic environment where RTT is constantly oscillating. As an example, due to the tremendous progress in physical and data layers reliability, in clear weather condition[4], a satellite link is considered as mostly error free (following DVB-S standard,

[4]In [23] the authors show that, for some rain events, the DVBS2 ACM modes can not cope with the deepest attenuations where the E_b/N_0 of the tropical rain is below the intervention threshold of the minimum ACM mode, which is -1.5 dB for a 10^{-7} QEF FEC (Quasi Error Free Packet Error Rate).

$BER \approx 10^{-7}$) while RTT might constantly moving due to mobility. Thus, we claim it would make sense, in certain condition, to prefer the use of rate-based mechanism rather than window-based mechanism as presented in Section I.

We reserve for a future work measurements dealing with vehicular networks. Indeed, current ns-2 simulator is not well designed either to implement correct MAC and physical layers models or to produce realistic VANET scenario. We are currently analysing various possibilities to realize this task and in particular investigating ns-2 extension software such as [24].

VI. Conclusion and Future Work

In this paper we have investigated and proposed a complete reliable rate-based protocol based on TFRC and SACK mechanisms. Our design also introduces a flow control variable, which regulates the sender to avoid packet loss at the receiver due to a slow receiver. We show that the modifications resulting from this composition does not affect the TCP-friendliness property of TFRC. We validate our proposal through ns-2.30 simulation and verify TCP-friendliness metrics. We further show that there is no packet loss due to flow control, at the receiver, and apply a stability criterion to demonstrate that the introduction of the flow control inside TFRC does not alter the smoothness property of this mechanism. We finally show the benefit of using this protocol in the context of long data transfer over a complex and realistic topology.

There is room for potential improvements. Following a recent study in [25], in a future work, we will investigate the benefit of tuning the number of feedback messages in order to adapt the frequency of feedback messages depending of the RTT, to guarantee optimal operation under various network conditions. Others experiments must be propose in order to verify hypothesis concerning the adequacy of such proposal over VANET and multihop scenario. We are currently specifying such scenarios based on real VANET traces in ns-2 and hope to obtain interesting results to discuss.

Acknowledgements

The authors would like to thank Yong-Han Lee for the implementation of this proposal during his internship done at NICTA and Eugen Dedu for the flower network settings. This work has been supported by funding from National ICT Australia.

References

[1] B. Ford and J. Iyengar, "Breaking up the transport logjam," in *in Seventh ACM Workshop on Hot Topics in Networks (HotNets-VII)*, Calgary, Alberta, Canada, Oct. 2008.

[2] L. Wood, W. Eddy, W. Ivancic, J. McKim, and C. Jackson, "Saratoga: a delay-tolerant networking convergence layer with efficient link utilization," in *IWSSC*, Salzburg, Austria, Sept. 2007.

[3] S. Farrell and V. Cahill, "Evaluating LTP-T: A DTN-Friendly transport protocol," in *IWSSC*, Salzburg, Austria, Sept. 2007.

[4] M. Handley, S. Floyd, J. Pahdye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," IETF, Request For Comments 3448, Jan. 2003.

[5] E. Kohler and M. Handley and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," IETF, Request For Comments 4340, Mar. 2006.

[6] V. Jacobson, "Congestion avoidance and control," in *In proceedings of ACM SIGCOMM*, Aug. 1988.

[7] S. Sharafkandi and N. Malouch, "Simple and Effective End-to-End Approach to Increase TCP Throughput over Ad-hoc Networks," in *The 19th International Teletraffic Congress*, Aug. 2005.

[8] V. Kawadia and P. R. Kumar, "Experimental Investigations into TCP Performance over Wireless Multihop Networks," in *In proceeding of the 2005 ACM SIGCOMM workshop on Experimental approaches to wireless network design and analysis*, Aug. 2005.

[9] K. Chen, K. Nahrstedt, and N. Vaidya, "The utility of explicit rate-based flow control in mobile ad hoc networks," in *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, Mar. 2004.

[10] G. Anastasia and A. Passarella, "Towards a novel transport protocol for ad hoc," in *Personal Wireless Communications (PWC)*, Sept. 2003.

[11] A. Leiggener and R. Schmitz and A. Festag and L. Eggert and W. Effelsberg, "Analysis of Path Characteristics and Transport Protocol Design in Vehicular Ad Hoc Networks," in *In Proceedings of the 63. IEEE Semiannual Vehicular Technology Conference*, May 2006.

[12] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, " TCP Selective Acknowledgment Options," IETF, Request For Comments 2018, Oct. 1996.

[13] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP," IETF, Request For Comments 2883, July 2000.

[14] G. Jourjon, E. Lochin, and P. Sénac, "Design, implementation and evaluation of a qos-aware transport protocol," *Elsevier Computer Communications*, vol. 31, 2008.

[15] L. Ma, X. Wu, and W. T. Ooi, "Tcp urel, a tcp option for unreliable streaming," School of Computing, National University of Singapore," Tech. Report, Oct. 2007, available on line: http://nemesys.comp.nus.edu.sg/projects/tcpurel/.

[16] S. Floyd and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control," Mar. 2006.

[17] E. Lochin, L. Dairaine, and G. Jourjon, "gTFRC, a TCP Friendly QoS-aware Rate Control for Diffserv Assured Service," *Springer Telecommunication Systems Journal*, vol. 33, no. 1, pp. 3–31, Dec. 2006.

[18] B. Briscoe, "Flow rate fairness: Dismantling a religion," ACM SIGCOMM CCR," Internal Report, Oct. 2006.

[19] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the Internet," *IEEE/ACM Transactions on Networking*, vol. 7, no. 4, pp. 458–472, 1999.

[20] J. Widmer, "Equation-Based Congestion Control," Diploma Thesis, University of Mannheim, Germany, Feb. 2000.

[21] D. X. W. C. Jin and S. H. Low, "FAST TCP: motivation, architecture, algorithms, performance," in *IEEE Infocom '04*, Hongkong, 2004.

[22] E. Dedu and E. Lochin, "A study on the benefit of tcp packet prioritisation," in *The 17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2009)*, Weimar, Germany, Feb. 2009.

[23] M. A. Vazquez and D. Pradas, "Voip cross-layer control for hybrid satellite wimax networks," *IEEE Wireless Communications Magazine special issue on âĂIJWireless Technologies Advance for Emergency Rural Communications*, June 2008.

[24] K.-c. L. Feliz Kristianto Karnadi, Zhi Hai Mo, "Rapid generation of realistic mobility models for vanet," in *IEEE WCNC*, 2007.

[25] G. Sarwar, R. Boreli, G. Jourjon, and E. Lochin, "Improvements in dccp congestion control for satellite links," *IEEE International Workshop on Satellite and Space Communications, 2008. IWSSC 2008*, Oct. 2008.