This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2007 proceedings.

Towards sender-based TFRC

Guillaume Jourjon
National ICT Australia Ltd, and
University of New South Wales and,
Université de Toulouse,
guillaume.jourjon@nicta.com.au

Emmanuel Lochin
National ICT Australia Ltd,
Australia,
emmanuel.lochin@nicta.com.au

Patrick Sénac
ENSICA-LAAS/CNRS and,
Université de Toulouse,
France,
senac@ensica.fr

***Abstract*—Pervasive communications are increasingly sent over mobile devices and personal digital assistants. This trend has been observed during the last football world cup where cellular phones service providers have measured a significant increase in multimedia traffic. To better carry multimedia traffic, the IETF standardized a new TCP Friendly Rate Control (TFRC) protocol. However, the current receiver-based TFRC design is not well suited to resource limited end systems. We propose a scheme to shift resource allocation and computation to the sender. This sender based approach led us to develop a new algorithm for loss notification and loss rate computation. We demonstrate the gain obtained in terms of memory requirements and CPU processing compared to the current design. Moreover this shifting solves security issues raised by classical TFRC implementations. We have implemented this new sender-based TFRC, named TFRC$_{light}$, and conducted measurements under real world conditions.**

I. Introduction

The recently standardized DCCP protocol [1] is seen as providing an efficient mechanism to carry multimedia traffic. DCCP can apply multiple congestion control mechanisms, and identifies TCP-Friendly Rate Control (TFRC) as congestion control ID #3 (DCCP/CCID3) [2]. TFRC is a congestion control mechanism for unicast flows operating in a best-effort Internet environment[3]. TFRC reproduces the TCP window-based congestion control mechanism through an equation model of the TCP equivalent throughput. The smooth rate variation, induced by this congestion control mechanism, makes it a good candidate for the delivery of an efficient transport service to client-server multimedia applications. However in such a media streaming scenario if multimedia servers are powerful processing and communication engines, this is not the case of mobile clients. Indeed, these clients are resource-limited end systems and are much more sensitive to communication and system processing while focusing on application layer.

Therefore the lightening of recurrent communication processing is a critical issue for increasing the performance and autonomy of mobile end systems. One of the main costs of the TFRC mechanism comes from the periodic computation of both the RTT and the loss rate of data carried by a connection. In particular, RFC 3448 [3] proposes the loss rate estimation to be done on the receiver side. It also suggests that this computation could also be done on the sender-side: *"It would be possible to implement a sender based variant of TFRC where the receiver uses reliable delivery to send information about packet losses and the sender computes the packet loss rate and the acceptable transmit rate"*.

We develop this idea by specifying and evaluating the design of a sender-based implementation of the TFRC congestion control mechanism. In our proposal, the reliable transfer of feedback packets is ensured by using a SACK-oriented mechanism [4]. This scheme is known to be robust to lossy channels while not entailing heavy and complex error control mechanisms [4]. Moreover, because it is located on the flows' servers only, the proposed sender-based approach is more robust to selfish receivers. Indeed, the sender no longer depends on the accuracy and the integrity of the returned information [3]. Some solutions to secure TFRC from selfish receiver have been proposed in [5] using RTSP [6]. Our solution requires fewer and simpler modifications to the TFRC header and algorithm than the proposal in [5].

Furthermore, a receiver-based solution achieves a periodic estimation of the loss event rate before sending it to the sender. This computation requires maintenance of a loss event history data structure. Such a receiver based solution does not comply with the capacities and resource constraints (i.e. in terms of energy consumption and overall computational performance) of light mobile receivers (e.g. PDAs, mobile phones) which are increasingly pervasive.

This paper is structured as follows: section II introduces the context of this study and provides some background information. Section III gives insights into the design of the new congestion control protocol architecture. Section IV compares the performance of the proposed congestion control protocol with respect to the standard TFRC implementation. We quantify the benefits of our proposal in terms of algorithmic processing and communication load in section V. Finally, section VI provides some conclusions and future directions.

II. Context and Related work

TFRC estimates the equivalent TCP sending rate X from equation (1) below. This equation depends on the mean packet size s and two periodically processed parameters: the packet loss event rate p and the round trip time RTT. RTO refers to the TCP retransmission timeout value which is usually a linear function of the RTT.

$$X = \frac{s}{(RTT \cdot \sqrt{\frac{p \cdot 2}{3}} + RTO \cdot \sqrt{\frac{p \cdot 27}{8}} \cdot p \cdot (1 + 32 \cdot p^2))} \quad (1)$$

1-4244-0353-7/07/$25.00 ©2007 IEEE

During the initialization phase, TFRC acts as TCP does during the slow start algorithm. This slow start phase also occurs during the transfer after the RTO timeout expires. This phase is followed by a congestion avoidance phase as soon as the receiver detects a loss. At this step, TFRC needs to estimate the loss rate in order to compute the sending rate X. The receiver evaluates the packet loss rate by a sliding window loss history structure. This structure stores the eight most recent loss event intervals. A loss event and its related interval of packets is defined as one or more lost packets during a duration of a least one RTT[3]. In other words, several packets lost during an RTT define a single loss event and the duration of a loss interval is greater than or equal to the RTT. The algorithm used at the receiver side is given in figure 1.

```
ReceivePacket() {
  Add packet to packet history;
  p_new = new value of packet loss rate;
  if (p_new > p_old){
    feedback timer expiration;
    do CreateFeedback();
  }
}
CreateFeedback() {
  compute average packet loss rate;
  calculate measured receive rate;
  prepare and send feedback packet;
  restart feedback timer;
}
```

Fig. 1. Original algorithm of the receiver

Two main issues can be identified in the receiver-based implementation algorithm. Firstly, the receiver must continuously maintain and update the loss event history data structure. The management of this data structure is an undesirable processing and memory management overhead for resource limited mobile receivers. Secondly, the receiver has to continuously process the loss event rate and send it to the sender, at least once per RTT, and as soon as it observes a loss event rate increase. Once again, this processing load squeezes the remaining processing capacity of the receiver. Moreover, such a receiver-based implementation cannot guarantee that selfish receivers do not try to trick the sender by inaccurately reporting the loss rate in an attempt to get higher bandwidth [5].

III. Design

This section presents the design of our sender-based TFRC protocol named TFRC$_{light}$. The design of this protocol is based on shifting the loss rate estimation processing to the sender side. We identify and propose the changes entailed by this shifting in the feedback packet structure and in the data structures managed by the receiver. The aim of our new TFRC protocol architecture and design is to reduce the receiver load. We discuss in this section the design of TFRC$_{light}$ by first presenting the underlying problems to the packet loss rate estimation shifting. Then, we give efficient solutions to these problems.

A. Notification of packet loss

In the original TFRC, the receiver has to periodically send feedback information to the sender. These feedback messages contain two parameters that allow the sender to estimate the current RTT value. These parameters are respectively (1) the timestamp of the last packet received (`Last Timestamp`) and (2) the amount of time elapsed between the receipt of the last packet and the generation of the feedback (`Processing Time`). We present these fields of the TFRC header in figure 2.

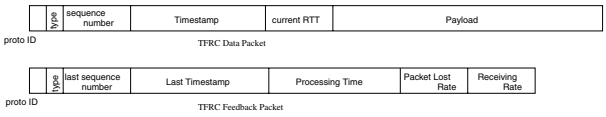


Fig. 2. Example of TFRC header

Moreover, feedback packets also contain information about the packet loss rate (`Packet Loss Rate`) and the received throughput (`Receiving Rate`) as processed by the receiver. In TFRC$_{light}$, the packet loss rate is no longer processed and returned by the receiver. Nevertheless, the receiver remains the only entity able to detect the loss of a packet and to notify the sender about this loss.

In order to perform this notification, we propose maintaining a compact data structure at the receiver. This data structure is a vector of bits (i.e. a SACK vector) that describes, from a given packet number, the distribution of packets received and lost. In other words, if a given packet is received, the bit is set to 0 otherwise 1. This vector is periodically sent.

The SACK vector offers redundancy that contributes to the reliable delivery of loss information when its sending interval is lower than the duration covered by the SACK vector. The value of the feedback packet sending period will be discussed in the next section. The right vector length can be chosen by considering that the sender-based and receiver-based implementation should behave similarly to packet losses. Indeed, as defined in [3], the sender no-feedback timer expires after $4 * RTT$. Where RTT, is the exponentially weighted moving average of the round trip time sent by the sender in each packet. A SACK-based mechanism is intrinsically robust to a maximum period of data losses equivalent to vector range. Then, the loss vector length should cover at least:

$$4 * RTT * PacketSendingRate$$

Where $PacketSendingRate$, is the sending rate included in each data packet header or computed by the receiver as the received packet rate. In order to reproduce the no-feedback timer behaviour of the standard receiver based version of TFRC, the loss information vector length must be dynamically recomputed with a period of $4 * RTT$.

The data structure used to compute SACK is a circular buffer, with a pointer keeping track of the most recently received packet. In the next section we first consider a simple initial scheme for managing this structure. Then, from the

issues raised by this scheme, we will propose a solution that conforms to the standard TFRC behaviour.

The message headers for the simple initial scheme are given in figure 3.

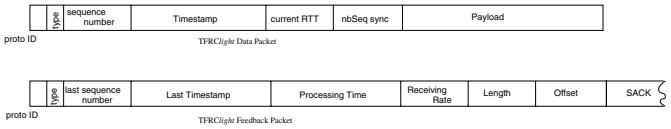


Fig. 3. Modification in TFRC header

B. Loss event definition in TFRC$_{light}$

Although the previously introduced data and protocol data unit structures are necessary for implementing an efficient sender-based TFRC protocol, they are not sufficient. Indeed, the loss history structure is based on the loss event definition given in [3]. A loss event is defined as the detection of one or more lost packets during one RTT. For keeping track of loss events, the receiver needs the receiving time of each packet to detect if lost packets correspond to the current loss event interval.

Since the sender and the receiver cannot maintain a synchronous behavior, the simple SACK structure previously introduced does not allow the sender to construct an accurate loss event history structure even if feedback packets are sent every RTT. Indeed, without a careful design, in certain cases, a loss event may be falsely detected. In figure 4, we give an illustration of such false detection. The time axis is used to represent the arrival time of the data packets. We also show on this axis the times, t_n, when the receiver sends feedback. As an example, we show the tail (i.e. the SACK vector) of three feedback messages below this axis. At t_1, the feedback message reports two losses represented by the two bits set in the `SACK` field. The `Offset` is equal to 100.

In the original TFRC, a timer of RTT time units should have been triggered at the estimated receiving time of the lost packet with the sequence number of 106. This timer range is represented in figure 4 by two-way arrows. At t_2, when the receiver sends its second feedback packet, the SACK vector `Offset` is now equal to 112 and as the RTT period is expired, a loss event should have been detected. At this time, the traditional TFRC algorithm closes the previous loss interval and restarts a new one from packet number 119. Finally at t_3, the losses reported for packets 125 and 127 belong to the previous loss event as the RTT timer expired at packet number 130. Since no other packet is lost after this expiration there is no new loss event. The problem of false detection can potentially result from an interpretation as a loss event of this third feedback with `Offset` field which is equal to 124 and its two marked bits in the vector.

As shown in figure 4, the TFRC mechanism is supposed to see two loss events (symbolized by the two RTTs). In TFRC$_{light}$, if we just shift the packet loss rate estimation, since there is no information about the estimated time of the packet loss, and the sender and receiver are not synchronous, the TFRC mechanism will see three loss events. Indeed, it will receive three disjoint feedback messages (one per RTT) with a non-null `SACK` field. Therefore, a simple logical interpretation of these feedbacks leads to the identification of three loss event instead of only two.

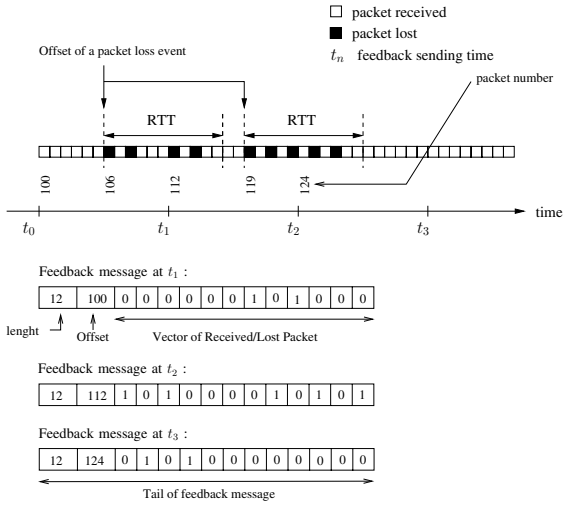
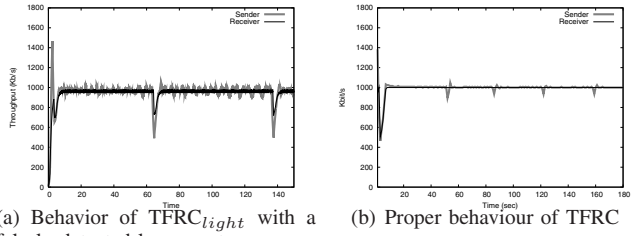


Fig. 4. Illustration of a the definition of the loss event

Figure 5 presents the impact of this false detection problem. We give in this figure the instantaneous throughput measured at the sender and instantaneous throughput measured at the receiver. Figure 5(a) shows the resulting throughputs of a TFRC$_{light}$ with a bad interpretation of loss events. The experiments involve an architecture with two nodes that generate traffic and are connected by a link with fixed capacity of $1Mbit/s$ and $RTT = 100ms$. In figure 5(a), TFRC$_{light}$ detects five loss events just after the slow start phase (between $t = [0, 10]$)[1]. However a correct implementation of TFRC would have seen only four loss events as illustrated in 5(b).



(a) Behavior of TFRC$_{light}$ with a falsely detected loss event

(b) Proper behaviour of TFRC

Fig. 5. Comparison of TFRC$_{light}$with a false detection and a usual TFRC in a network with a bandwidth of $1Mbit/s$, and an RTT=$100ms$

As a result, when a new loss event occurs (i.e. $t = 63s$ and $t = 137s$), the sender will decrease its emission rate more than needed. In figure 5(a), this behaviour can be seen with the two rate dips. This throughput decrease is explained by the way the loss history structure is built. Indeed, as the mechanism

[1]Observed by the addition of a memory variable inside the core protocol

gets successive loss events, the corresponding entries in the loss history structure will be filled with loss intervals shorter than they should be. When a new loss event occurs, these erroneously sized loss intervals raise the resulting value of the loss event rate. This loss rate causes an excessive reduction of the sending rate as given by equation (1).

In order to solve this issue we propose the following modifications.

1) New receiver algorithm: At the receiver side the structure remains similar to the one presented in the previous section. The algorithm used by the receiver side is shown in figure 6.

```
ReceivePacket(){
  Add packet to received packet;
}
CreateFeedback(){
  calculate measured receive rate;
  prepare and send feedback packet;
  restart feedback timer;
}
```

Fig. 6. Receiver algorithm

In this proposal, the receiver is no longer responsible for computing the packet loss rate. This algorithm supposes the existence of a new structure that records the arrival or loss of packets.

2) Modification at the sender side: In order to detect a loss event at the sender side, the server has to set up a structure that stores information about when packets were sent. This structure is identical to the one that traditional receiver-based TFRC receivers use to compute the packet loss rate, except that instead of keeping a trace of the arrival time of the packet this new structure stores the sending time of the packet.

Based on this new structure the sender is now able to detect loss events from a sender perspective by considering the sending time of the packets reported as lost in the received SACK vectors. Furthermore because the sender keeps the packets sending time, the `TimeStamp` field in both data and feedback headers is no longer needed. Fig. 7 gives the resulting new structure of the TFRC$_{light}$ headers associated with the data and feedback packets.

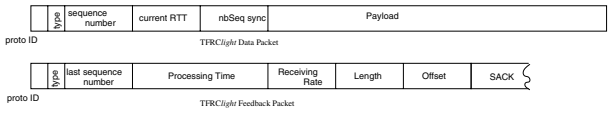


Fig. 7. Modification in TFRC header for the loss event detection second solution

3) Translation from Loss History to Loss Events: In our proposal, the sender is now aware of the sending time of each packet. This information, combined with the received SACK vectors, allows the sender to process the packet loss rate as detailed in Fig. 8.

In section 5.2 of RFC 3448, the authors explain how to build loss events from the loss history. This operation needs:

```
for(int i=0; i<lenghtACK; i++)
{
  if(vector[i]==0)
    add Packet(offset+i) loss History;
    p_new=new value of packet loss rate;
  else
    loss history to loss event;
}
compute average packet loss rate;
```

Fig. 8. Analysis of the vector of Ack

- S_{loss} the sequence number of the lost packet;
- S_{before} the sequence number of the last packet to arrive such that $S_{before} < S_{loss}$;
- S_{after} the sequence number of the first packet to arrive such that $S_{loss} < S_{after}$;
- T_{before} the reception time of S_{before};
- T_{after} the reception time of S_{after}.

In the presented solution, the sender is not aware of T_{before} and T_{after}. Nevertheless, the sender must estimate the arrival time of S_{loss}. In our proposal, we use sending times, not arrival times, to build loss events. These sending times are corrected by the following factor, which the sender evaluates whenever it receives feedback (where X_{sent} and X_{recv} are respectively the sending and receiving rates):

$$\alpha = \frac{X_{sent}}{X_{recv}}$$

The determination of the new event is accomplished in the same way as in the original TFRC except that the time reference is no longer the arrival time but is now the sending corrected by the factor α.

4) Discussion: As feedback messages are not systematically sent when a loss is detected, we recommend that the feedback message sending interval should equal $\frac{RTT}{x}$ with $x > 1$.

IV. Validation of TFRC$_{light}$

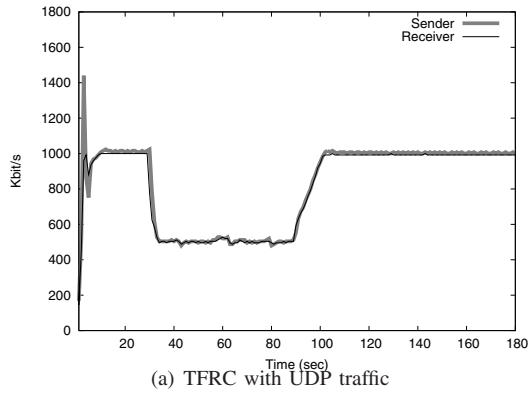
We have implemented a user level prototype of TFRC$_{light}$ in Java. We have evaluated the TFRC$_{light}$ protocol over a simple testbed composed of two end-systems and a network emulated by a FreeBSD/Dummynet pipe [7]. For all experiments, the bandwidth and the RTT are respectively set to $1Mbit/s$ and $100ms$. In all figures, we report the sending/receiving instantaneous throughputs measured respectively at the sender/receiver sides. The results of our experiments show that our sender based protocol has the same behaviour as traditional receiver based TFRC implementations.

We made many measurements to validate this new architectural design and report in this section a representative sample of the results.

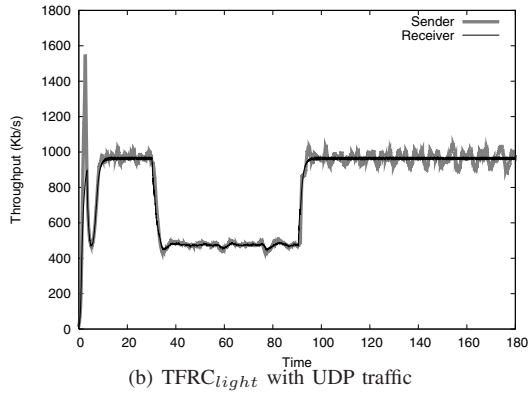
It is always difficult to compare the performance of a real implementation and a simulated one since the simulation reproduces an ideal case without the overhead introduced by real measurements. Nevertheless, we show that the TFRC$_{light}$ receiver throughput is as stable as the ns-2 receiver throughput.

Concerning the sender throughput, more oscillations occur in TFRC$_{light}$ than in ns-2 TFRC. This is explained by the overhead introduced by our user level TFRC$_{light}$implementation.

In the experiment illustrated in figure 9, we introduce an UDP flow with a rate of $500Kbits/s$ between $t = [30sec, 90sec]$. This test aims to verify the responsiveness of TFRC$_{light}$ compared to ns-2 TFRC. In figure 9, due to the packets being multiplexed with a non-responsive UDP flow, both implementations brutally decrease during the UDP flood. Furthermore, both implementations react the same way to the losses induced by the UDP flow. When the UDP flow stops, both implementations respond similarly. Eventually, we can conclude from this scenario that the modifications proposed and implemented in TFRC$_{light}$ result in a behaviour similar to ns-2 TFRC.



(a) TFRC with UDP traffic



(b) TFRC$_{light}$ with UDP traffic

Fig. 9. TFRC and TFRC$_{light}$ with a network bandwidth of $1Mbit/s$, an RTT=$100ms$ and introduction of an UDP flow at $t = [30s, 90s]$

In the following experiments, we show that the proposed sender-based TFRC remains TCP friendly. The results of the TFRC friendliness property are shown in table IV. These measurements give the average throughput observed at the receiver side after $200s$ of transfer. We drive the first experiment with 5 TFRC$_{light}$ flows only. We have also studied the multiplexing behaviour of TFRC$_{light}$ flows with TCP and TFRC flows. The results summarized in Table 1 show that TFRC$_{light}$ flows occupy a fair share of the bandwidth when multiplexed with TCP and TFRC flows.

	TFRC$_{light}$	TCP	TFRC
5TFRC$_{light}$	19.4%(20%)	N/A	N/A
5TFRC$_{light}$ and 5TFRC	10%(10%)	N/A	9.5%(10%)
5 TFRC$_{light}$ and 10TCP	6.2%(6.6%)	6.7%(6.6%)	N/A

TABLE I
MEAN BANDWIDTH RATIO PER FLOW (THEORETICAL VALUE IN BRACKETS)

V. QUANTIFICATION OF THE SHIFTING SCHEME

In table II, we summarize the benefits/drawbacks of the proposed design compared to the original algorithm.

benefits	suppression of the loss history structure no processing of the packet loss rate protection from misbehaving receivers simpler timer management simpler sender's algorithm
drawbacks	new structure for Sack vectors management Loss events built from sender point of view feedback only sent periodically

TABLE II
SUMMARY OF THE BENEFITS AND DRAWBACKS OF THIS PROPOSAL

The main advantages of our solution are the removal of the packet history structure and the removal of the computation of the packet loss rate at the receiver. Conversely, we have introduced a new light structure that allows the receiver to build the Sack vector sent to the sender in feedback messages. This structure has a size of the order of $4RTT * Bandwidth/(packetsize)$. For instance, in the case of a transmission with a bandwidth of $1Mbit/s$, an RTT of $100ms$ and a packet size of $1000Bytes$, the structure should have a maximum size of $50bits$. This structure is actualized for each data packet received. In the original design of TFRC, the receiver has to manage a more complex structure that stores information concerning the arrived or lost packets. The stored information includes:

- the packet timestamp (16bits);
- the packet size (8bits);
- the arrival time (16bits).

Therefore, the elementary size of an entry is $40bits$. Furthermore the size of this structure does not have a maximum bound. This structure is emptied after detecting a loss event. As an example in figure 5, there are no losses between $t = 63$ and $t = 137$. During this entire period, the structure has to be updated at a rate of $1Mbit/s$ which corresponds to $125packet/s$. This structure for the given example contains:

$$40 * 125 * (137 - 63) = 370Kbits$$

when it can be released. In this particular case the memory use decreases from $370Kbits$ to $50bits$ with TFRC$_{light}$.

To estimate the computation benefit of our proposal, consider how in normal TFRC [3] the loss rate estimate is supposed to be recomputed for every received packet. The basic

algorithmic sequence for computing the loss rate estimate entails the following set of elementary arithmetic operations: eight additions, eight multiplications, one division and one maximum operation. For instance, at rate of $1Mbit/s$ with a packet size of $1Kbyte$, this estimation should be computed 125 times per second. These elementary operations can be translated into CPU cycles as follows[2]:

- division = 70 cycles
- multiplication = 15 cycles
- addition, maximum = 0.5 cycles

As a result, for the given example, in the original TFRC, the receiver has to use $24312.5\ cycles/s$.

Furthermore, after a slow start phase the receiver has to initiate its loss history. This initialization is done from the inversion of equation (1) in order to find the packet loss rate corresponding to the measured received rate. This initialization is usually done with a binary search and uses the list of elementary operation sum up in table III.

	$+$	$*$	$/$	$sqrt$
binary search	$4n+4$	$8n+8$	$2n+2$	n
CPU cycles	0.5	15	70	70

TABLE III
LIST OF THE NUMBER OF ELEMENTARY OPERATION
($n = number\ of\ iterations$)

The worst case of this binary search can be observed when this algorithm diverges, which can occur when the solution of the inversion of (1) is outside the $[0, 1]$ range. This potential of divergence leads to an upper bound on the number of iterations done during the binary search. Therefore, in order to compute the inversion of (1) for most cases, the maximum number of iterations is usually set to 50. Indeed, we implemented the binary search of the inversion and found out that the algorithm still converges in 15 iterations for $RTT = 400ms$ and $bandwidth = 1Mbit/s$.

In conclusion, for the worst case it takes 16862 CPU cycles for the initialization process. In our proposal, all of this computational process is achieved at the sender side. Moreover, we have shown in section IV that this simplification entails a congestion control behaviour that strictly conforms to receiver-based TFRC implementations.

VI. Conclusion and future work

In this paper, we have presented the design of a sender-based TFRC congestion control. This design is driven by the aim to shift the computation of the loss rate estimation from the receiver to the sender in order to alleviate the processing and memory needs of "light" receivers. This shifting requires the sending of loss resilient feedback, and is accomplished through the use of a SACK-like mechanism. The result is a significantly lightened computational load on the receiver which is particularly useful for mobile clients with computation and energy constraints. We have shown that a sender-based TFRC built following the proposed protocol architecture and mechanisms behaves the same as the official ns-2 implementation and remains friendly to TCP streams. We have quantified the benefits of this shifting from a computational and memory point of view. Furthermore, the solution proposed allows the security issues raised in [3] to be solved. These security issues are related to the forwarding of false loss event rates by the receiver. Such misbehaviour is no longer possible with our solution when associated with nonce mechanisms. We plan to further validate our proposal by performing a large range of experimental measurements on a multi-hop testbed.

[2]According to Intel PIV documentation

VII. Acknowledgments

We would like to thank Tim Moors from the University of New South Wales for his valuable remarks. This work has been supported by National ICT Australia funding.

References

[1] E. Kohler and M. Handley and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," IETF, Request For Comments 4340, Mar. 2006.
[2] S. Floyd, E. Kohler, and J. Pahdye, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)," IETF, Request For Comments 4342, Mar. 2006.
[3] M. Handley, S. Floyd, J. Pahdye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," IETF, Request For Comments 3448, Jan. 2003.
[4] 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.
[5] M. Georg and S. Gorinsky, "Protecting tfrc from a selfish receiver," in *Proc. of Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services (ICAS/ICNS 2005)*, Oct. 2005.
[6] H. Schulzrinne, A. Rao, and R. Lanphier, "Real Time Streaming Protocol (RTSP)," IETF, Request For Comments 2326, Apr. 1998.
[7] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," *ACM Computer Communications Review*, vol. 27, no. 1, Jan. 1997.