Design, implementation and evaluation of a QoS-aware transport protocol

Guillaume Jourjon [a,b,c,*], Emmanuel Lochin [a], Patrick Sénac [c]

[a] National ICT Australia Ltd., Australia
[b] University of New South Wales, Australia
[c] CNRS/LAAS – ENSICA, France

Abstract

In the context of a reconfigurable transport protocol framework, we propose a QoS-aware Transport Protocol (QSTP), specifically designed to operate over QoS-enabled networks with bandwidth guarantee. QSTP combines QoS-aware TFRC congestion control mechanism, which takes into account the network-level bandwidth reservations, with a Selective ACKnowledgment (SACK) mechanism in order to provide a QoS-aware transport service that fill the gap between QoS enabled network services and QoS constraint applications. We have developed a prototype of this protocol in the user-space and conducted a large range of measurements to evaluate this proposal under various network conditions. Our results show that QSTP allows applications to reach their negotiated QoS over bandwidth guaranteed networks, such as DiffServ/AF network, where TCP fails. This protocol appears to be the first reliable protocol especially designed for QoS network architectures with bandwidth guarantee.

Keywords: Transport protocol; TFRC; SACK; QoS networks

1. Introduction

Specifications of transport layer protocols have traditionally considered only basic QoS parameters such as reliability and order constraints. Some solutions to provide QoS-oriented information to the application, such as the Real-time Transport Protocol (RTP) [1], have been proposed but these solutions, above the transport layer, require an application support for QoS control because they do not provide any explicit QoS control mechanisms. Following the increase of end-systems and network performances, new application QoS requirements have emerged and induce new QoS constraints in underlying communication services, such as delay or bandwidth requirements. These new requirements drive an evolution toward QoS awareness of the two fundamental layers of any communication architecture: the network and transport layers. At the network layer level, many architectures have been proposed such as IntServ, DiffServ or MPLS [2–4]. In particular, the DiffServ architecture provides two classes of service, the first one, Expedited Forwarding (EF), offers bounds for the delay and the jitter. The second class, Assured Forwarding (AF), provides a high delivery probability as long as the aggregated traffic does not exceed the negotiated rate (also called target rate). The DiffServ/AF class is well suited for continuous multimedia streaming applications as well as bulk data transfer, which need some level of bandwidth guarantee and are resilient to packets losses. Nevertheless, these kinds of network services do not address the whole set of applications requirements (e.g. reliability, order).

As a result, a transport protocol aims at complementing the underlying network service. The use of TCP in this context is still very popular. However, TCP is designed for

* Corresponding author. Address: National ICT Australia Ltd., NPC, Locked Bag 9013, Alexandria, NSW 1435, Australia. Tel.: +61 2 83745594.

E-mail addresses: guillaume.jourjon@nicta.com.au (G. Jourjon), emmanuel.lochin@nicta.com.au (E. Lochin), senac@ensica.fr (P. Sénac).

applications that, in the context of the classical best-effort (BE) Internet service, need reliable and ordered packets delivery without considering other QoS constraints (e.g. time constraints). Several studies have shown that TCP does not fit well with network-level QoS services [5]. Some new transport layer mechanisms or protocols have been recently introduced in order to deliver a transport service more compliant with multimedia application QoS constraints. In particular, the TCP Friendly Rate Control mechanism (TFRC) [6] entails smoother rate variations than the AIMD based TCP congestion control mechanism. Nevertheless, neither TCP nor TFRC take into account the QoS guarantee offered at the network-level by the DiffServ/AF service.

Therefore, to date, there is no transport protocol able to take into account consistently both multimedia applications needs and the QoS offered by an underlying QoS network infrastructure. This paper aims to fill that gap and to deliver an efficient mapping between application QoS needs and QoS network services while enforcing flows' TCP friendliness of their out-profile part (i.e. best-effort). The new QoS-aware Transport Protocol (QSTP) introduced in this paper is implemented in a reconfigurable transport framework [7], that makes possible to combine dynamically several transport mechanism (or micro-protocols) for delivering the transport service that best fits to the application needs and to the underlying network services. QSTP results from the specialization of the TFRC congestion control mechanism and its composition with a Selective Acknowledgment error control mechanism.

The present contribution also aims at demonstrating how the combined use of a TFRC specialization (guaranteed TFRC) and SACK can improve a TCP compliant transport service, especially during losses bursts. Indeed, these mechanisms share the common goal of improving the QoS delivered to flows by offering, respectively, a mechanism for enhancing flows' rate smoothness and a mechanism for loss recovery. Their combined use offers potential performance improvements that this paper aims at exploring. Therefore, we show how two QoS parameters, i.e. bandwidth and reliability, can be managed jointly in a non-conflicting way (i.e. conversely to TCP) for delivering a better transport service than TCP, regardless the underlying class based network service. In addition, the composition of the SACK and TFRC has two other main advantages: first, SACK allows fully or partially reliable error control disciplines to be achieved; second, the SACK information can be easily integrated within TFRC feedback packets.

In order to illustrate the benefit of using QSTP over bandwidth guaranteed network services, we show that QSTP is able to deliver throughput guarantee either on top of the DiffServ/AF class of service or on top of a generic guaranteed bandwidth network service.

This paper is structured as follows: Section 2 briefly presents related work about transport protocol over DiffServ/AF class. Then, Section 3 introduces the context of this study and provides some background information about the mechanisms used in QSTP. Section 4 presents in details the design of QSTP protocol and Section 5 is dedicated to the performance evaluation of the proposed protocol. Finally, Section 6 provides some conclusions and future directions.

2. Related work

The DiffServ/AF class has been specifically designed for elastic traffic such as the TCP traffic. Nevertheless, guaranteeing a minimum throughput to a TCP flow associated to this class of service is not feasible under certain network conditions [5]. In order to cope with this problem, many research works have focused on efficient TCP traffic conditioning. Unfortunately, the numerous proposed solutions [8–11] are still much sensitive to the network conditions and sometimes difficult to implement. In summary, we can say that TCP is not able to efficiently map its transport service toward network layer AF differentiated service without specific conditioning scheme. Moreover, TCP conditioning is complex since the timescales used, respectively, by the network and transport layer are different (i.e. the network uses a packet timescale and TCP uses an Round Trip Time (RTT) timescale). Finally, the parameters that allow a good flow conditioning to be achieved are hard to evaluate at the network level. Indeed, loss probability and RTT are difficult to measure in a passive manner at the edge of the network.

The main potential source of discrepancy between TCP and the AF service results from the TCP congestion control mechanism. Indeed, following packet losses, the TCP congestion control mechanism strongly reduces the sending rate and is totally oblivious of the rate guarantee offered by the underlying network service. Therefore, we have previously proposed the use of a QoS-aware congestion control mechanism to solve this issue [12].

3. Context

This study is achieved in the framework of the EuQoS project[1] funded by the 6th framework European research program. The EuQoS project aims at designing and experimenting scalable multi-domain communication architecture for the global delivery of QoS-centred communication services. In the context of the EuQoS project, we have investigated the concept of adaptive transport architecture simultaneously aware of the QoS application needs and underlying network services. The resulting adaptive transport architecture aims to be configured from a set of fundamental transport layer mechanisms (i.e. congestion, rate, error, order, or even time control) for applying the most efficient adaptation between the application needs and the available network services. We detail in the follow-

[1] http://www.euqos.eu/.

ing of this paper a specific instance of this generic transport framework which results from the composition of two specific mechanisms, that are, a congestion control (i.e. TFRC) and an error control mechanisms (i.e. SACK).

3.1. The TFRC congestion control

A TFRC sender [6] estimates its TCP equivalent sending rate X from Eq. (1) which takes as parameters the mean packet size s and two periodically processed parameters, the packet loss event rate p, and the round trip time RTT. In this equation RTO refers to the retransmission timeout value.

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

During the initialization phase, TFRC acts as TCP during the slow start algorithm. This slow start phase can also occur during the transfer if the RTO timeout expires. This phase is followed by a congestion avoidance phase, driven by Eq. (1), as soon as the receiver detects a loss. During the congestion avoidance phase, TFRC needs an estimation of the loss event rate in order to compute the sending rate, X. The packet loss rate is evaluated at the receiver side with the help of a sliding window based structure that maintains a history of loss events [6].

3.2. SACK mechanism

The concept of Selective ACKnowledgments (SACK) was originally introduced in [13] as a TCP option that aims to optimize its fully reliable service by allowing faster recovery of bursts of packet losses [14]. By sending selective acknowledgments, the receiver of data can inform the sender about which segments or packets have been successfully received and which ones have to be selectively retransmitted. On the other hand, SACK can make easier the design of a partially reliable transport service in accordance with the application data units importance [15].

4. QSTP design and implementation

This section presents the core mechanisms used to build the QSTP protocol. Firstly, the QoS-aware TFRC specialisation mechanism (gTFRC) is presented. Secondly, we will show how reliability is performed through an adaptation of SACK to gTFRC.

4.1. gTFRC

In the DiffServ/AF service class, the throughput of a flow is divided into two parts. The first one is a fixed part which corresponds to a minimum assured throughput; packets belonging to this part are marked in-profile. The second one is an elastic part which corresponds to an opportunist flow of packets marked out-profile. In the event of network congestion, the in-profile packets are preserved from losses. At the contrary, out-profile packets are conveyed on a best-effort principle and are dropped first if congestion occurs.

In case of excess network bandwidth, the application can send more than its target rate (i.e. more than its in-profile part), according to this policy, in this case the network has to mark out-of-profile the excess traffic. Conversely, when the network becomes congested, out-of-profile packets losses occur and the resulting loss rate estimated by TFRC, that integrates both in profile and out profile packets, can fall down bellow the target rate requested by the application. TCP would react in the same situation by halving its congestion window. As for TCP over the AF class [5], the TFRC mechanism is not aware that the loss is operated on out-profile packets and that it should not decrease its actual sending rate below the target rate. Concerning TCP, solutions proposed in [10,11] introduce a conditioner able to better mark the TCP flows by taking into account their sporadic nature. As TFRC explicitly computes the actual sending rate thanks to Eq. (1), gTFRC directly constraints this resulting rate to avoid the under-utilization of the allocated network bandwidth. The aim of this TFRC specialization consists in making the sending rate estimator aware of the target rate. This scheme avoids the cost of traffic conditioners while enhancing efficiently performances in terms of application throughput and TCP-friendliness.

In the TFRC standard algorithm, when the loss event rate p is not nil, the update of the sending rate X is basically computed as a minimum between the rate computed by the TFRC equation and two times the estimated rate of the receiver.[2] Our proposed TFRC specialization to the AF service consists in enforcing the TFRC rate estimation to be always higher than the target rate as follows:

$$X = \min(\max(X_{\mathrm{calc}}, g), 2 * X_{\mathrm{recv}}) \quad (2)$$

where X is the updated transmit rate in bytes/s, g is the target rate in bytes/s, X_{calc} is the rate in bytes/s computed from Eq. (1) and X_{recv} the estimated received rate. This mechanism has been thoroughly evaluated through ns-2 simulation and implementation. Further details about gTFRC measurements and design are, respectively, available in [12,16].

4.2. Reliable gTFRC

The previous section focuses on the first component of our QoS-aware reliable transport protocol. Indeed, gTFRC allows the target rate negotiated by the application to be insured while being TCP friendly. The next step is the integration of gTFRC with a SACK-based mechanism to provide a reliable transport service. We have seen in Section 3 that SACK offers a powerful foundation to provide a sophisticated error control mechanism much more effi-

[2] In any cases a minimum rate of one packet every 64 s is insured.

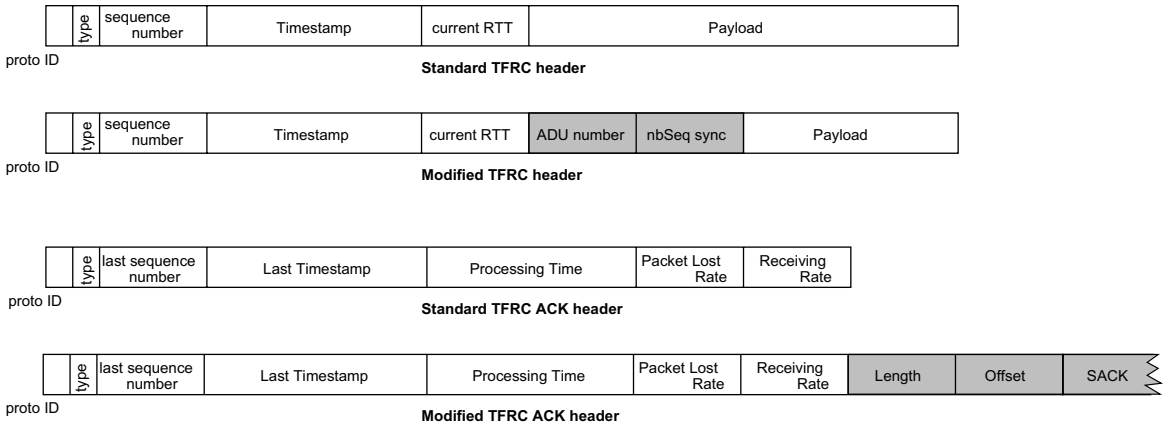


Fig. 1. Modification in TFRC header.

cient than the basic *Go back N* error recovery mechanism even in its TCP variant. As specified in [14], the SACK mechanism aims to return information about the set of missing TPDU.[3] Since TFRC is a datagram oriented mechanism and SACK is byte stream oriented, we adapt SACK to a datagram oriented transport service.

In Fig. 1, the two first protocol data units represent, respectively, the TFRC header and the new header that results from the composition of *g*TFRC and SACK. The two last PDU represent, respectively, the feedback given by the receiver for the classical TFRC protocol and TFRC/SACK composition. In these headers, each field is either 4 or 8 bytes encoded field except for the `proto ID` (one byte), the `type` (one byte) and the `SACK payload` (variable length). The datagram oriented SACK mechanism is defined in the same way as the stream oriented one. The `SACK payload` is constituted by a sequence of pairs of sequence numbers.[4] These pairs represent the edge of intervals of correctly received contiguous packet. The `length` represents the number of pairs to analyse for the sender. Finally the `Offset` represents the sequence number of the first packet of the first pair. We can note that the SACK mechanism can help to implement a partial order transport service that would retransmit mandatory packets only.

4.3. Discussion about the composition of the SACK and gTFRC mechanisms

In our proposal, the application has to provide the target rate negotiated with the QoS network to the transport protocol. This is done at the socket level through a `setsockopt()` function. Such an approach could potentially allow the application to abuse the network by giving a higher value than the guarantee *g*. Another potential issue is the case where the QoS service provider gives a wrong configuration to the application and the edge router. In the following sections, we tackle both problems.

4.3.1. Preserving the provider interest against a denial of service

As we give the possibility to instantiate through a `setsockopt()` function the target rate negotiated between the network service provider and the user, we can easily envisage that a misbehaving user could take part of feature by giving to *g* a higher value than the negotiated one.

In the context of a DiffServ/AF class, the edge router marks in-profile the packets according to the negotiated profile and out-profile the excess part. A misbehaving client will increase its out-profile traffic part and when a network congestion occurs, the dropping precedence set in the core router will entail the dropping of this excess traffic. Therefore, the misbehaving application will increase its own packet loss rate and will not get any bandwidth advantage. In summary, increasing the value of g at the user level does not impact on the in-profile traffic that is bounded by the SLA between the service user and the provider. Therefore, this kind of denial of service is avoided by the DiffServ conditioning mechanisms.

4.3.2. Preserving the network service user and provider against wrong network configurations

This second case can potentially induce issues both for the network service user and provider. Indeed, in this case a discrepancy between the user and provider configurations either would induce a risk for the service user to get a poorer service than the negotiated one or for the service provider a risk to dedicate to the service user more resources than needed. For instance, such an inconsistency could occur if the service provider miscalculates the resource needed for the related service layer agreement. In a DiffServ context, the in-profile traffic is not guaranteed anymore when a QSTP flows gets losses while emitting below its target rate. In such a case, two actions are possible for the sender. The first one is to pursue to emit at the guaranteed rate, *g*. This is a legitimate behavior since the service provider must provide to his client the service he has paid for. The second type of action would be to react to the observed congestion and to warn the application or the user that the SLA has been broken off. This can

[3] Transport Protocol Data Unit.
[4] This SACK structure could also be implement as a bit field.

be done thanks to an additional mechanism that would be able to detect that a bunch of losses occurred in the in-profile part. Anyway, in the case of an under-provisioned network, TCP (and TFRC) would react as if the target rate is lower than the expected one.

The TFRC algorithm prevents in a certain manner these problems. Indeed, the algorithm will not return a sending rate higher than twice the receiving rate (given by $2 * X_{\text{recv}}$ in (2)). However, we believe that these security concerns are out of the transport layer scope. We claim that it is definitely not the responsibility of the protocol to detect a selfish user behaviour or to react to a wrong setting. We therefore do not present results concerning an under-provisioning network.

4.4. Implementation

In this section we present the implementation of QSTP protocol based on a compositional transport protocol framework [7]. Basically, this framework, developed in Java language, allows to easily instantiate transport layer mechanisms and to compose them to build a transport protocol which applies an efficient adaptation between application needs and underlying network characteristics [7]. Fig. 2 gives an overview of the micro-protocols (i.e. processing modules) that have been composed for the instantiation of the QSTP protocol. QSTP is composed on both sides by seven Processing Modules (PM), respectively, dedicated to (see Fig. 2(a) for details):

- the processing of the outgoing flow (`Add Header`, `Set Header-Rate Ctrl` and `Send Sock`);
- the processing of the ingoing flow (`Remove Header`, `Process IN`, `Receive Sock`);
- the `Process Feedback` and the `Create Feedback` deal with the management of the feedback messages (i.e. creation and analysis).

The main components of QSTP are:

- the `Process IN` component: this component implements, at the sender side, the *g*TFRC mechanism;
- the buffer `Application Buffer IN`: this buffer is the transmission queue upstream the rate control component, packets to retransmit are placed on top of this queue;
- the buffer `Retransmission Buffer`: this buffer stores sent data but not yet acknowledged;
- the `Process Feedback` component: this component is in charge of the processing of feedback messages. This component applies error control on packets stored in the `Retransmission Buffer`;
- the `Create Feedback` component: this component computes the loss event rate and creates the Feedback message with the SACK structure.

Detailed descriptions of this framework can be found in [7].

5. Performance evaluation of QSTP

This section evaluates the QSTP service over a bandwidth guarantee networks. We firstly present the experimentation model used and the general hypothesis. Then, the results and their analysis are provided over various network conditions. In a sake of comparison, the chosen parameters are those used in other well-known papers about TCP over AF such as [5,8–10].

5.1. Model and general hypothesis

QSTP is implemented in Java language and evaluated over the DiffServ topology presented in Fig. 3. All the nodes are PC, the end-hosts run GNU/Linux and the routers run FreeBSD with ALTQ [17] in order to implement the DiffServ service and at the core router dummynet [18] is used to emulated the network. The experiments have been carried out using the following configuration:

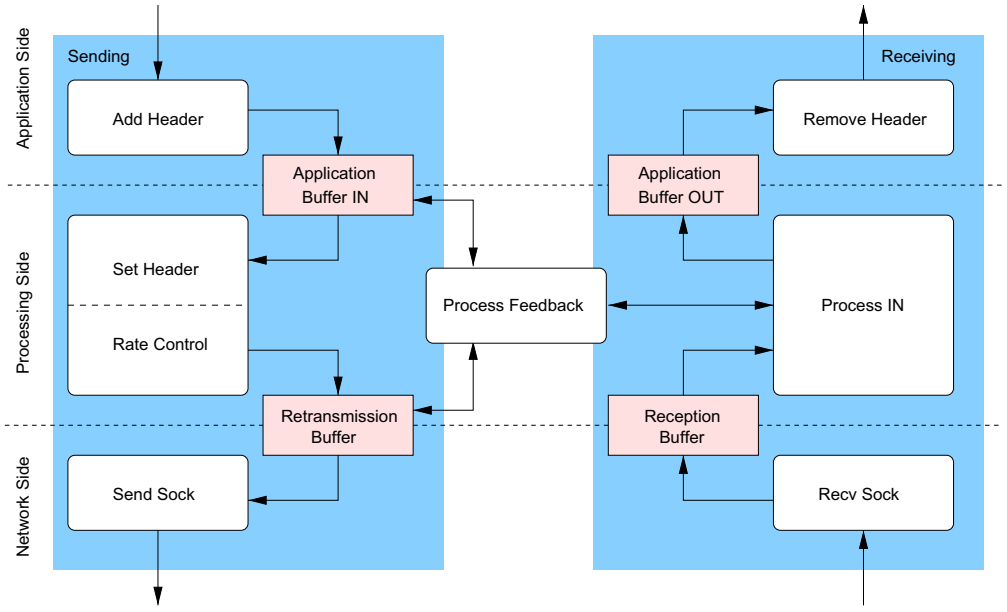
- the packet size is fixed to 1500 bytes;
- a two-colour token bucket marker with a bucket size of 10^4 bytes is used on the edge router [19];
- routers are configured with a queue size of 50 packets and RIO[5] parameters in the core router correspond to $(\min_{\text{out}}, \max_{\text{out}}, p_{\text{out}}, \min_{\text{in}}, \max_{\text{in}}, p_{\text{in}}) = (10, 20, 0.1, 20, 40, 0.02)$;
- the bottleneck between the core and the egress router has a fixed capacity of 1000 Kbits/s;
- measurements are carried out 10 times during 180 s for an FTP-like transfer.

We made experiments with a large set of different initial Round Trip Time delays (i.e. minimum measured RTT) and target rates. Only a representative part of these results are given in the next section. The choice of these results has been made since the various scenarii presented represent some of the worst cases for a unique flow (TCP and TFRC) to reach its target rate. In the following section we measure in a first time the throughput obtained at the network level at the receiver side. Then we present the "goodput" which measure the throughput at the application level. Finally, we present the jitter obtained for TCP and QSTP flow.

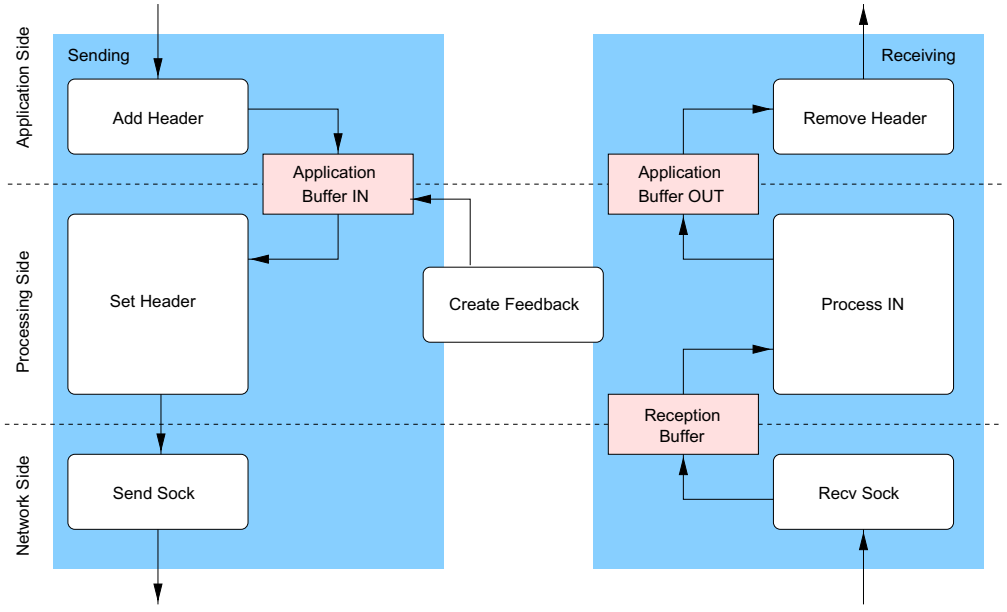
5.2. Analysis of the QSTP behaviour over a standard DiffServ/AF network scenario

This section aims at illustrating the QSTP behaviour above a DiffServ service. The measurements presented in Fig. 4 gives the corresponding instantaneous throughput at the network level on the receiver side. This throughput is computed using a time-sliding window algorithm of one second as explained in [20].

[5] RED In Out queue.



(a) Sender side



(b) Receiver side

Fig. 2. Internal mechanisms of the protocol at the sender and receiver side.

In this first experiment, we analyse the behaviour of one flow (i.e. a TCP, TFRC, or QSTP flow) versus a TCP aggregate of 15 micro-flows.[6] As QSTP provides an end-to-end per-flow guarantee, the aim of this first experiment is to verify whether QSTP is able to maintain the target rate negotiated whatever the network load. All the flows have an RTT of 30 ms. This flow has a target rate of 500 Kbits/s and crosses the (A,B) path of the DiffServ test-bed while the TCP flows aggregate has a target rate of 300 Kbit/s and crosses the (C,D) path. In all experiments, the TCP aggregate has always outperformed its target rate. In Fig. 4, we only report the results for the flow alone against the TCP aggregate. First, we give the result obtained by a TCP flow in Fig. 4(a). As explained in [5], the TCP flow is not in the best condition to reach its target rate since it has the highest target rate. Moreover, because of the TCP multiplexing behaviour, when two aggregates with a different number of micro-flows are in a network,

[6] We define as a microflow, a single instance of an application-to-application flow of packets which is identified by source address, destination address, protocol id, and source port, destination port (where applicable) following [21]. And we define an aggregate a set of two or more microflows.

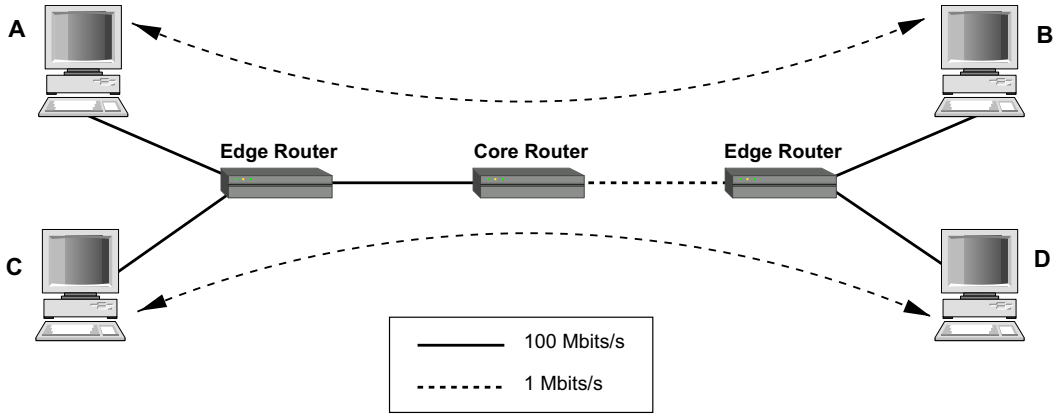
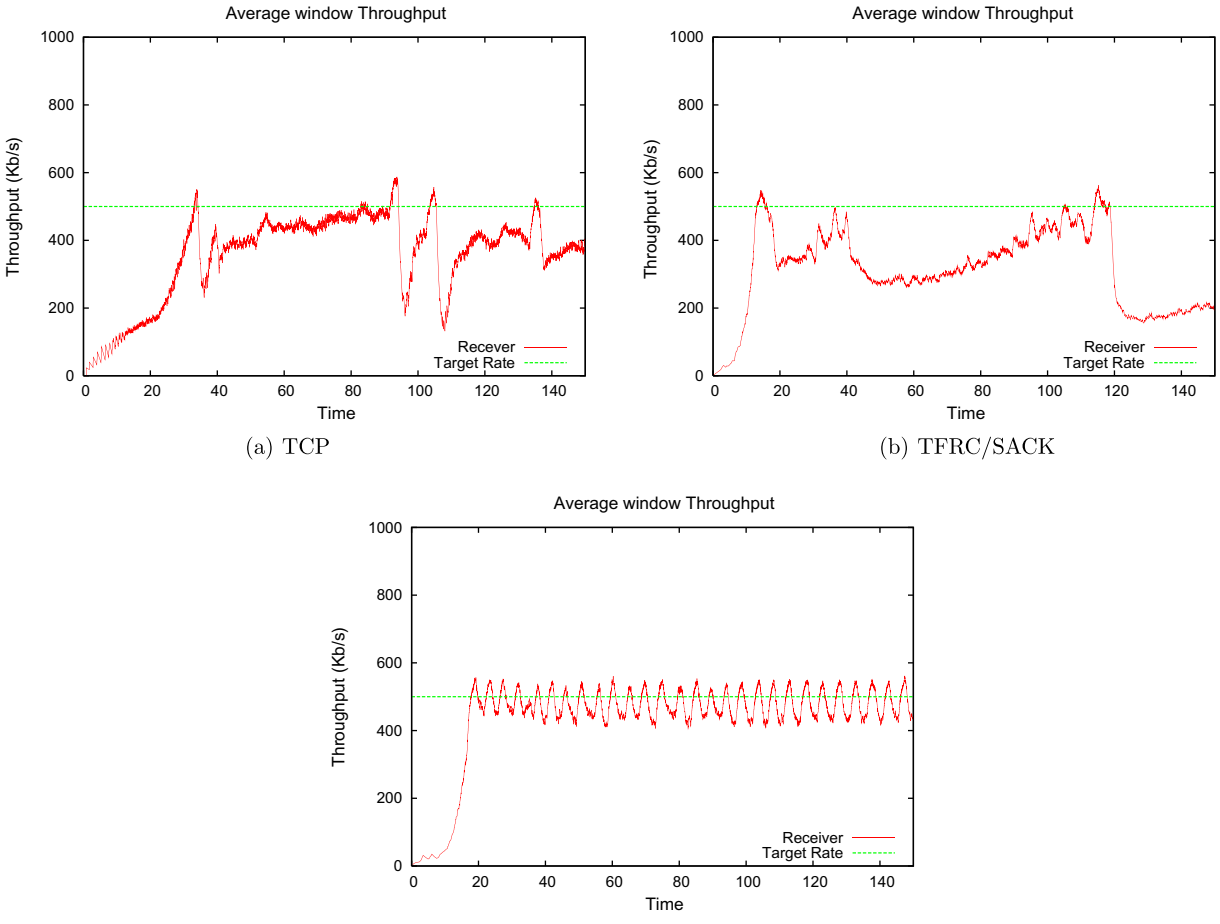


Fig. 3. The testbed topology for DiffServ experiments.



(a) TCP

(b) TFRC/SACK

(c) QSTP: gTFRC/SACK

Fig. 4. Throughput of one TCP, TFRC/SACK, QSTP flow versus a 15 TCP flows aggregate.

the larger outperforms the smaller [5]. Fig. 4 shows that the TCP flow does not reach its target rate.

In the next Fig. 4(b), we give the result obtained for a TFRC/SACK flow multiplexed with the same 15 micro-flows aggregate. In this experiment, TFRC/ SACK does not reach its target rate either. Since TFRC reproduces the TCP window congestion control behaviour and since we have added a reliability mechanism, we could expect to obtain a behaviour almost similar to TCP on average. Nevertheless, the smoothing TFRC property makes the TFRC/SACK flow less aggressive than the TCP ones. As the bottleneck of the network becomes loaded, the RTT and the losses in the network increase. As a result, we can see between $t = [40\text{ s},\ 100\text{ s}]$ that TFRC mechanism recovers slowly to the after a transient congestion [22].

To cope with the unawareness problem, the QSTP protocol composes gTFRC and SACK mechanisms. The results depicted in Fig. 4(c) illustrates that, conversely to the TCP and TFRC flows, the QSTP flow is able to achieve the requested target rate. As a conclusion, thanks to the composition of these two mechanisms, QSTP can be considered as a DiffServ/AF compliant reliable protocol. Indeed, we only use for these experiments standardized and implemented DiffServ mechanisms such as the token bucket two-colour marker on the edge and the RIO queue on the core.

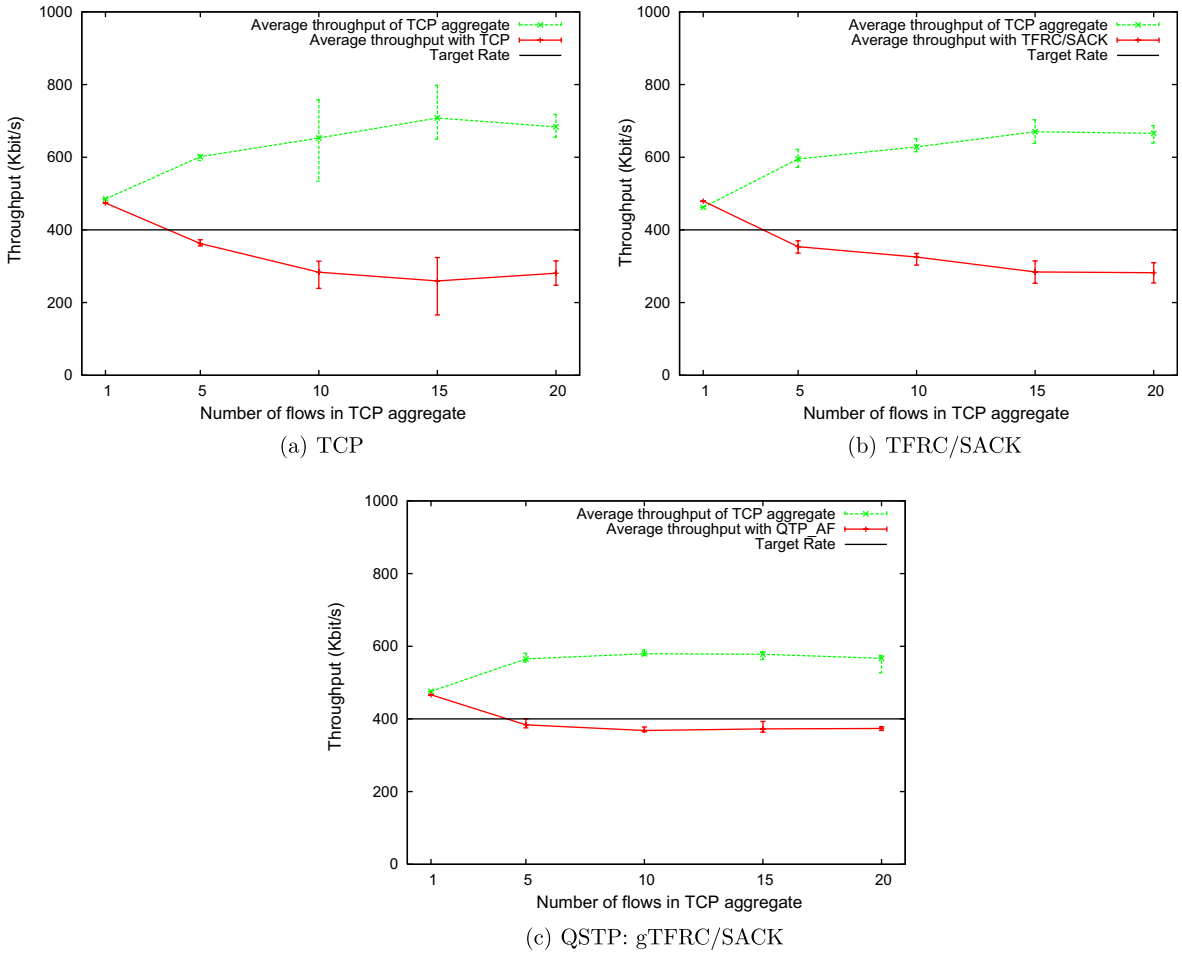
The next section will focus on the study of the impact of these three transport services on the QoS offered to the application layer (i.e. the transport service user). In this context, measurements focus on the application throughput (or goodput) at receiver side. In case of a FTP transfer, it corresponds to the throughput data transfer.

5.3. Impact of QoS perceived at the user level

In this study, one flow (*from host A to host B*) is in competition with a variable size aggregate. The aggregate (*from host C to host D*) has a variable number of micro-flows ranging from 1 to 20. The RTT of all flows is set to 30 ms and target rates of (A, B) and (C, D) are equal to 400 Kbits/s. Fig. 5 gives the results obtained for TCP, TFRC/SACK and QSTP flow versus the variable aggregate.

We report in Fig. 5 the average throughput of the single flow and the average throughput of the aggregate (both computed after 150 s) with the min/max values of ten consecutive measurements As already evaluated in a DiffServ network [5], Fig. 5(a) illustrates that TCP flow does not reach its target rate. Concerning the TFRC/SACK composition, Fig. 5(b) shows that the distance between the throughput variation amplitude is inferior to the one of TCP. This is due to the smoother property of TFRC congestion control. Nevertheless, on average, the obtained throughput is in the same order of magnitude than TCP. Finally, Fig. 5(c) confirms the previous results, showing that the QSTP flow (A,B) reaches the requested target rate no matter the number of micro-flows in competition in the (C,D) aggregate.

Note that the difference between the target rate at the network level and the throughput delivered at the user level is simply due to the QSTP/IP protocol overhead. More-



(a) TCP

(b) TFRC/SACK

(c) QSTP: gTFRC/SACK

Fig. 5. Average throughput according to the number of micro-flows in the aggregate.

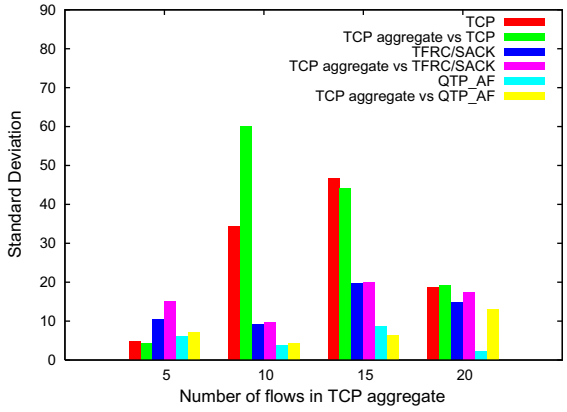
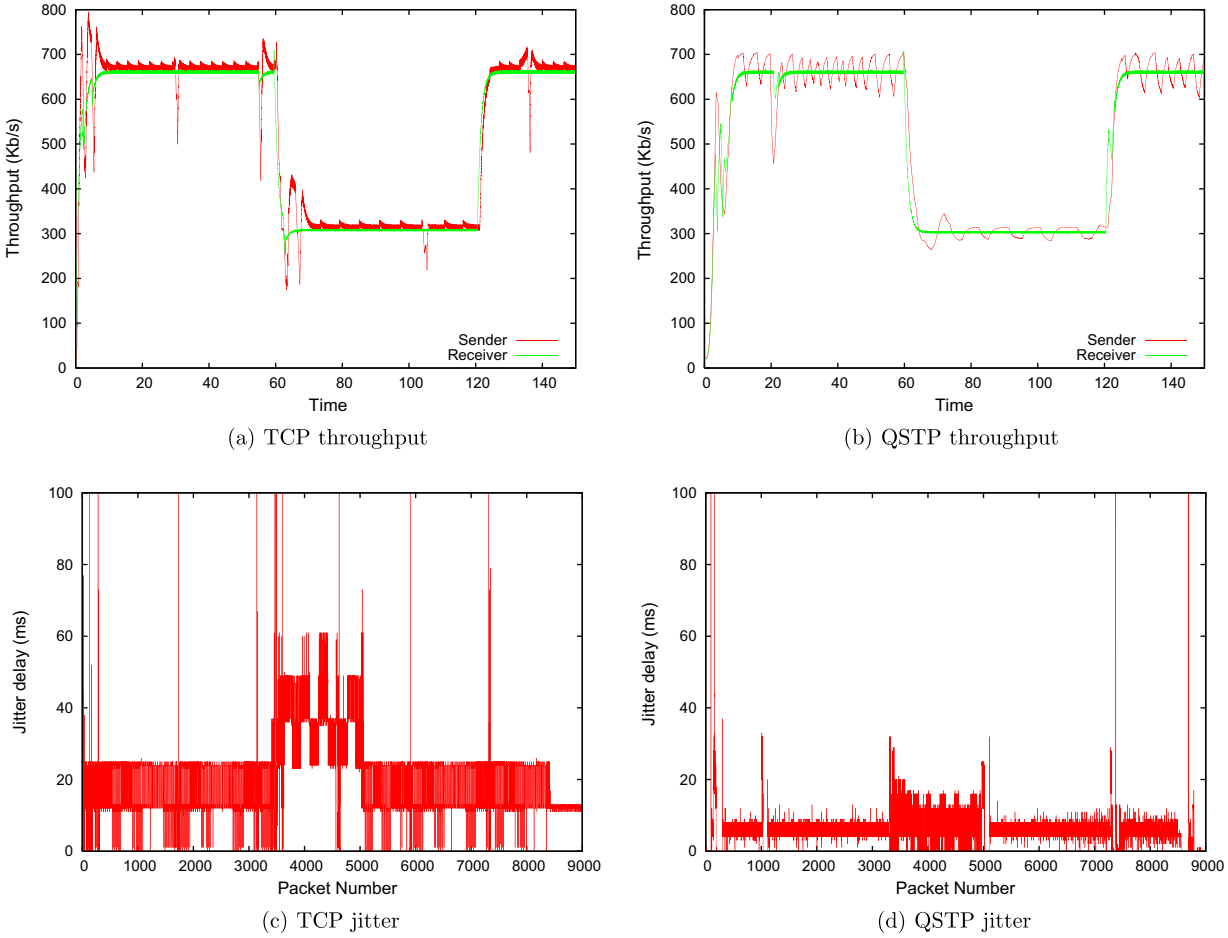


Fig. 6. Throughput standard deviation.

over, the min/max interval is the smallest one. The measurement overhead at the application level explains the gap between the target rate line (expressed at network level) and the QSTP average application throughput in Fig. 5(c). For the sake of accuracy and in order to quantify the min/max values, we give separately in Fig. 6 the standard deviation of these results. This figure confirms the stability of TFRC and *g*TFRC over a differentiated network. Indeed, we can see that the standard deviation for these two congestion controls mechanism is small. The non-compliant TCP behaviour with a DiffServ network is highlighted by its large standard deviation.

5.4. Illustration over a QoS network with a bandwidth guarantee

In this section, we focus on the behaviour of QSTP on top of another network level QoS mechanism. This allows us to verify that the proposed protocol can be used over any kind of network providing a bandwidth guarantee. To perform this evaluation, we configure a QoS network with a Class Based Queueing (CBQ) scheduling mechanism [23] that provides a guaranteed pipe of 300 Kbit/s for the studied flow (i.e. TCP or QSTP). The network topology used in these experiments remains identical to the one presented in Fig. 3. The emulated QoS network does not use any admission control. The CBQ is configured in "borrow mode". It means that in case of non-congestion, the BE traffic can borrow bandwidth into the reserved pipe. This case of configuration is more general as this kind of scheduling algorithm is currently available in commercial routers such as CISCO 4000 and above series. Fig. 7(a) and (b)



(a) TCP throughput

(b) QSTP throughput

(c) TCP jitter

(d) QSTP jitter

Fig. 7. Jitter of one TCP, QSTP flow versus a UDP flow with various throughputs.

show, respectively, the throughput of TCP and QSTP at the sender and receiver side. Fig. 7(c) and (d) show the jitter of these two flows. In these experiments, both flows compete with an UDP flow.

During the experiment, the UDP flow emits at 300 Kbit/s except between [60,120] s where it emits at 1000 Kbit/s. As a result, during this interval the bottleneck is full. Fig. 7(a) and (b) give the throughput measured at the sender and receiver side. Once the congestion occurs, the CBQ algorithm starts (i.e. when the UDP flow sends above 700 Kbit/s). Thanks to the CBQ scheduling, both flows obtain their guarantee as shown in these figures.

Fig. 7(c) and (d) give the network level jitter in milliseconds obtained by both flows. We can see that TCP 7(c) obtains a higher jitter than QSTP 7(d). This is an expected result as TFRC congestion control algorithm has the property to emit a non-bursty traffic. In an obvious way, the resulting jitter must be lower. However, these graphs show that the composition of TFRC with SACK does not impact on this standard behaviour and the resulting jitter for QSTP is lower than for TCP.

6. Conclusion and future works

In this paper, we have presented the design of a QoS transport protocol based on TFRC congestion control and SACK mechanisms. This proposal has been implemented and evaluated in the context of a compositional transport protocol framework. QSTP proposes a transport service that results from the composition of a reliability mechanism with a QoS-aware congestion control mechanism. Measurements show that this composition defines the first reliable transport protocol compliant with DiffServ/AF class. In particular, we show that applications by using QSTP obtain their negotiated target rate with a small standard deviation under various network conditions.

The next step of this work is the deployment and the performance evaluation of QSTP over a large scale European QoS aware network designed in the framework of the EuQoS European project. We expect a large range of measurements in order to complete this study with real network conditions and various DiffServ/AF classes. We are currently starting a standardization process of this protocol at the IETF. A first draft concerning the *g*TFRC congestion control is under revision [16]. We expect to integrate the full definition of the presented protocol into the next draft version.

Acknowledgments

This research work has been conducted in the framework of the EuQoS European project (http://www.euqos.eu). This work has been supported by funding from National ICT Australia. The authors are also grateful Dr. Sebastien Ardon and Dr. Laurent Dairaine and Dr. Ernesto Exposito for their useful comments.

References

[1] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, RTP: A Transport Protocol for Real-Time Applications, Request for Comments 3550, IETF, Jul. 2003.
[2] R. Braden., D. Clark, S. Shenker, Integrated Services in the Internet Architecture: An Overview, Request for Comments 1633, IETF, Jun. 1994.
[3] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, Assured Forwarding PHB Group, Request for Comments 2597, IETF, Jun. 1999.
[4] E. Rosen, A. Viswanathan, R. Callon, Multiprotocol Label Switching Architecture, Request for Comments 3031, IETF, Jan. 2001.
[5] N. Seddigh, B. Nandy, P. Pieda, Bandwidth Assurance Issues for TCP Flows in a Differentiated Services Network, in: Proc. of IEEE GLOBECOM, Rio De Janeiro, Brazil, 1999, p. 6.
[6] M. Handley, S. Floyd, J. Pahdye, J. Widmer, TCP Friendly Rate Control (TFRC): Protocol Specification, Request for Comments 3448, IETF, Jan. 2003.
[7] E. Exposito, Specification and Implementation of a QoS Oriented Transport Protocol for Multimedia Applications, PhD Thesis, LAAS-CNRS/ENSICA, Dec. 2003.
[8] X. Chang, J.K. Muppala, On improving bandwidth assurance in AF-based DiffServ networks using a control theoric approach, Computer Networks 49 (6) (2005) 816–839.
[9] B. Nandy, P. Pieda, J. Ethridge, Intelligent traffic conditioners for assured forwarding based differentiated services networks, in: IFIP High Performance Networking, Paris, France, 2000.
[10] M. El-Gendy, K. Shin, Assured forwarding fairness using equation-based packet marking and packet separation, Computer Networks 41 (4) (2002) 435–450.
[11] E. Lochin, P. Anelli, S. Fdida, Penalty shaper to enforce assured service for TCP flows, in: IFIP Networking, Waterloo, Canada, 2005.
[12] E. Lochin, L. Dairaine, G. Jourjon, gTFRC: a QoS-aware Congestion Control Algorithm, in: Proc of the 5th International Conference on Networking, Mauritius, 2006.
[13] V. Jacobson, R. Braden, TCP Extensions for Long-Delay Paths, Request for Comments 1072, IETF, Oct. 1988.
[14] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, TCP Selective Acknowledgment Options, Request for Comments 2018, IETF, Oct. 1996.
[15] Ernesto Exposito, Michel Diaz, Patrick Sénac, Design principles of a QoS-oriented Transport Protocol, in: IFIP International Conference on Intelligence in Communication Systems, Bangkok, 2004.
[16] E. Lochin, L. Dairaine, G. Jourjon, Guaranteed TCP Friendly Rate Control (gTFRC) for DiffServ/AF Network, Internet Draft draft-lochin-ietf-tsvwg-gtfrc-01, IETF, Jun. 2006.
[17] K. Cho, Managing Traffic with ALTQ, in: Proceedings of USENIX Annual Technical Conference: FREENIX Track.
[18] L. Rizzo, Dummynet: a simple approach to the evaluation of network protocols, in: ACM Computer Communications Review 27 (1) (1997).
[19] J. Heinanen, R. Guerin, A Single Rate Three Color Marker, Request for Comments 2697, IETF, Sep. 1999.
[20] W. Fang, N. Seddigh, A.L., A Time Sliding Window Three Colour Marker, Request for Comments 2859, IETF, Jun. 2000.
[21] K. Nichols, S. Blake, F. Baker, D. Black, Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, Request for Comments 2474, IETF, Dec. 1998.
[22] J. Widmer, Equation-Based Congestion Control, Diploma Thesis, University of Mannheim, Germany, Feb. 2000.
[23] S. Floyd, V. Jacobson, Link-sharing and resource management models for packet networks, IEEE/ACM Transactions on Networking 3 (4) (1995) 365–386.