European Commission — FP7

R/TE

REDUCING INTERNET TRANSPORT LATENCY

Project acronym: **RITE**

Project number: 317700

Work package: End systems and applications

Deliverable number and name:

Deliverable 1.1: End-system analysis and preliminary development report

Title: End-system analysis and preliminary development report
Work Package: WP1

Version: 2
Date: November 1. 2013
Pages: 30

Author:
Andreas Petlund

Co-Author(s):
Iffat Ahmed, Anna Brunstrom, Per Hurtig, Gorry Fairhurst, Raffaello Secchi, Carsten Griwodz, David Hayes, David Ros, Ing-Jyh Tsang, Michael Welzl, Olga Bondarenko, Minoo Kargar

To:
Rüdiger Martin
Project Officer

Status:
[ ] Draft
[ ] To be reviewed
[ ] Proposal
[X] Final / Released to CEC

Confidentiality:
[X] PU — Public
[ ] PP — Restricted to other programme participants
[ ] RE — Restricted to a group
[ ] CO — Confidential

Revision:
(Dates, Reviewers, Comments)

Contents:
Report describing the outcome of the end-system analysis. The most promising avenues for further investigation and prototype development are identified. The report also describes initial results on simulations and prototype developments of Task 1.2 and Task 1.3

Abbreviations

This section provides definitions of key terms and defines the abbreviations used in the remainder of the report.

3G 3rd Generation

ACK Acknowledgement

ADU Application Data Unit

API Application Programming Interface

AQM Active Queue Management

BE Best Effort

BoF Birds-of-a-Feather

CAIDA Cooperative Association for Internet Data Analysis

CC Congestion Control

CM Congestion Manager

CMT-SCTP Concurrent Multipath Transfer for SCTP

CDF Cumulative Density Function

CDN Content Delivery Network

cwnd Congestion WiNDow

CWV Congestion Window Validation

DASH Dynamic Adaptive Streaming over HTTP

FIFO First-In First-Out

FSE Flow State Exchange

GB Gigabyte

HAS HTTP Adaptive Streaming

HTTP HyperText Transfer Protocol

IETF Internet Engineering Task Force

ISP Internet Service Provider

LEDBAT Low Extra Delay BAckground Transport

LKM Loadable Kernel Module

LTE Long Term Evolution

MB Megabyte

MPTCP Multipath TCP

NAT Network Address Translation

NVP Non Validated Period

NS-2 Network Simulator 2

OWD One Way Delay

PDV Packet Delay Variation

PSP PipeACK Sampling Period

RFC Request For Comments

RITE Reducing Internet Transport Latency End-to-End

RMCAT RTP Media Congestion Avoidance Techniques (IETF Working Group)

RT Real-Time

RTCWEB Real Time Collaboration on world wide WEB (IETF Working Group)

RTO Retransmission Time Out

RTP Real-Time Protocol

RTT Round Trip Time

RW Restart Window

SBD Shared Bottleneck Detection

SCTP Stream Control Transmission Protocol

SVD Singular Value Decomposition

TFRC TCP Friendly Rate Control

TCP Transmission Control Protocol

TCPM TCP Maintenance (IETF Working Group)

TSQ TCP Small Queue

UDP User Datagram Protocol

WiFi Wireless Fidelity

WG Working Group

WP Work Package

Participant organisation name	**Participant Short Name**	**Country**
Simula Research Laboratory	SRL	Norway
BT	BT	UK
Alcatel-Lucent	ALU	Belgium
University of Oslo	UiO	Norway
Karlstad University	KaU	Sweden
Institut Mines-Télécom	IMT	France
University of Aberdeen	UoA	UK

1 Introduction

End systems can contribute significant sources of latency to the communication due to non optimised protocol design, interactions within the protocol stack or local buffering. They can also incur latency by making purely local decisions that lead to interactions with other traffic on bottlenecks routers, in particular by making incorrect assumptions about other traffic's behaviour. Applications with latency requirements such as the chosen use-case applications in RITE (financial applications, online games and interactive video) depend on timely delivery. On the side of the end user, this timely communication must happen in a diverse environment of competing traffic. Considering this, much can be done to optimise end-host operations to better support time-dependent communication. In the analysis phase of RITE, we have explored the potential of latency-reducing mechanisms to address the objectives of work package 1.

The task summarised in this deliverable has focused on finding the sources for latency in the end systems. After reviewing the state-of-the-art and exploring the most promising areas of investigation, we have chosen some focus areas where we expect our efforts to achieve the maximum latency reduction. Some of these focus areas have matured into into preliminary results.

The analysis phase has also uncovered problem spaces that are less promising now than at the time the proposal was written. In such cases, an evaluation of whether RITE should continue in this direction or channel its resources towards more promising topics has been made.

1.1 Analysis breakdown and document structure

This document presents a summary of the preliminary analysis phase in Work Package 1 of the RITE project. The main focus of this report is on the issues that can be addressed in the end-host without active interaction with the network.

The following are the key contributions of work package 1:

- We have analysed transport latency from an end-host perspective to identify where the efforts of the RITE project will have the greatest impact.
- We have completed preliminary work for the most promising avenues of investigation resulting in a detailed map of the topics that should be prioritised for the rest of the RITE project.
- We have performed a thorough survey of the state-of-the-art for the topics of investigation. This survey will be published as a journal article, but the key findings are listed in this report.

More specifically, this document presents:

- An analysis of how flow completion time is limited by the start-up latency and how this problem becomes a severe limitation as the majority of connections on the Internet are short flows (§ 2).
- An evaluation of the value of continuing in RITE the work on reducing buffers in the operating systems in view of recent initiatives from other projects (§ 3).
- A proposal for improving recovery latency for application limited TCP and SCTP streams by modifying the retransmission timeout (RTO) mechanism (§ 4.1).
- An analysis of how problems related to capacity sharing affects latency and how competing traffic may negatively impact the latency of time-dependent flows (§ 5).
- A proposal for better estimation of what the congestion control window size should be for bursty traffic with long idle intervals in order not to overshoot the window while still gaining a proper start window at the beginning of a new burst (§ 5.1).

- A proposal for a mechanism to better share information between flows with the same origin traversing common bottlenecks in order to more intelligently share the bottleneck capacity between the coordinated flows (§ 5.2).
- An analysis of how to detect bottlenecks shared between flows in order to dynamically adapt latency-reducing mechanisms for flows that share a bottleneck (§ 5.3).
- An analysis of the effect of current transport APIs on time-dependent traffic and an initiative to standardise APIs for choosing and negotiating the best transport service for latency-sensitive applications (§ 6).
- An analysis of how multiple paths can be utilised to improve latency with a recommendation on future efforts in that direction for the RITE project (§ 7)

This deliverable is closely related to deliverable 2.1 ("Network systems analysis and preliminary development report") that is submitted at the same time. Whereas this deliverable addresses work package 1 and has a focus on end-host mechanisms, deliverable 2.1 addresses network mechanisms and interactions between end-hosts and the network. Some of the subprojects include some elements that belong to work package 1 and other elements belonging to work package 2. To avoid redundancy, we have chosen to describe each subproject in only one of the deliverables. We have tried to make it clear from the text how different elements relate to the research areas of the different work packages. The documents should therefore be read in conjunction. Deliverable 3.1 ("Traffic pattern analysis and data set acquisition report") may also be a useful reference when it comes to questions of trace usage and testbeds for RITE subprojects.

The structure of the remainder of this report is as follows: Section 2 analyses how flow completion time can be reduced by improving the start-up latency. Section 3 gives an analysis of buffer latency issues for end-systems. Section 4 provides an analysis of protocol specific delays, whereas section 5 describes the analysis of capacity sharing mechanisms. In Section 6, API latency issues are examined and discussed. Section 7 discusses multipath transport protocols in the context of low latency. A summary of the main results of our analysis is given in Section 8. Finally, Section 9 concludes this report and describes the key findings.

2 Scaling Transport Dynamics and Start-up Delays

Internet transport protocols have to be designed on the basis that each time a flow starts the available capacity is unknown. Even a re-start after idling cannot assume that the capacity or other traffic has remained unchanged. So senders universally use some variant of the TCP slow-start algorithm, which sends an initial handful of packets, waits for feedback, then doubles how much it sends in each subsequent round as long as it has sensed no losses in the feedback. This problem is not unique to TCP—in the absence of any knowledge about the available capacity all data flows face the same dilemma.

TCP's exponential slow-start is traditionally considered an acceptable compromise between acceleration and overshoot. However, our analysis of the problem shows that slow-start does not scale. Every doubling of bottleneck capacity has added another round trip of delay before slow-start can exploit it. More capacity only improves performance for flows larger than a limit that doubles as fast as link capacities double. And, even for flows that are large enough to exploit faster links, the amount of loss experienced during overshoot doubles. Below we outline the key parts of our analysis, discuss the implications of our findings, and describe our work plan for RITE within this area.

2.1 Quantifying the Problem

To analyse the problem we model the average transfer rate of a single TCP flow with a dedicated bottleneck capacity. The general approach is to find the number of round trips in which slow-start only partially fills the bottleneck, *before* the round in which either slow-start ends or the flow ends, if sooner. Then any remaining packets will arrive at the bottleneck rate, whether they are all sent in the next round while still in slow-start phase or there are enough packets to progress into the fast retransmit and congestion avoidance phases. Any handshaking rounds are excluded by the assumption that either the flow is re-starting after idle or using TCP fast open after an earlier connection between the same hosts.

Average transfer rate rather than completion time is chosen as the 'figure of merit' a) in order to normalise with respect to transfer size and b) to emphasise how much potential there is to use, rather than waste, link capacity.

To quantify how bad slow-start is, we want to ensure that we criticise the best possible variant of slow-start. Therefore we use state of the art parameters in the modelling, that is:

- Initial window of ten Ethernet-size segments [1], as well as three for comparison [2];
- No use of multiple parallel flows (because a larger initial window is equivalent);
- A range of round trip times that are realistic for the public Internet, with and without caching;
- Base-two exponential increase (i.e. doubling per round);

For a wide range of sizes of each transfer (1B to 1GB), Figure 2.1 shows the average rate achieved on a log-log scale for a few scenarios. Taking the second plot in Figure 2.1 as an example (IW=10, R=20ms, X=80Mb/s), it shows that unless the transfer size is larger than around 1MB, it hardly even starts to exploit a dedicated 80Mb/s link. A 1MB transfer can only average about 40Mb/s, while a smaller 15kB transfer can only average about 10Mb/s.

For transfer sizes less than $\approx$ 500kB, any capacity increase beyond about 80Mb/s would make no noticeable difference to average rate (illustrated by comparing the X=80Mb/s and the X=1Gb/s plot in Figure 2.1). A 2Mb/s plot is also included to show that it is much more useful to many more people to increase capacity from 2Mb/s to 80Mb/s, because it considerably increases the average rate of a broad range of popular sizes of transfer (any larger than $\approx$ 10kB).

Finally, the lowest two plots of Figure 2.1 show that the problem is significantly worse if the round trip time is longer. For an inter-continental round trip time that is ten times longer (R=200ms) only flows ten times larger (more than $\approx$ 10MB) can start to make use of 80Mb/s capacity.

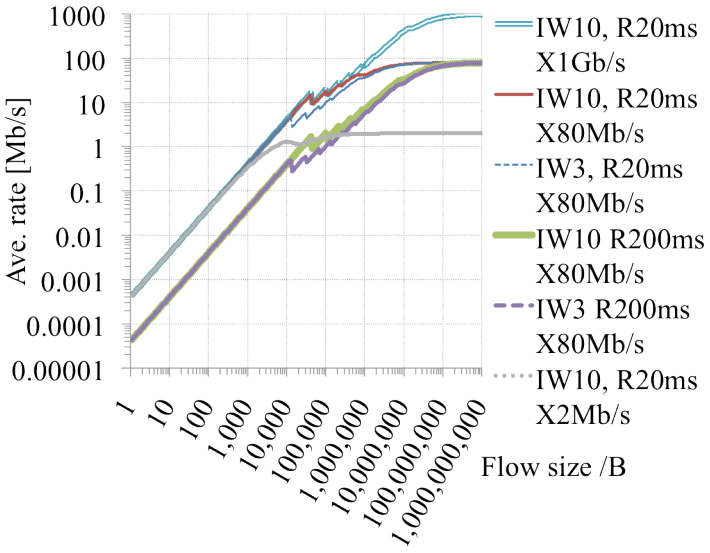

Figure 2.1: The problem: TCP has to limit the average transfer rate of different size flows despite dedicated capacity X. IW=initial window; R=round trip time;

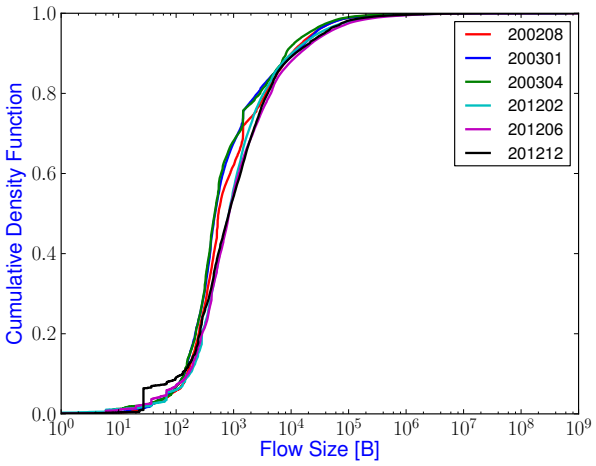

Figure 2.2: Prevalence of flows of different sizes on the current Internet, based on traces from CAIDA [3]

The scaling behaviour illustrated in Figure 2.1 would not be a problem if all transfer sizes were getting larger, but they are not. As we continue to invest in greater link capacity, larger transfers become feasible. But it does not follow that all the demand for the smaller files disappears—the range of feasible transfer sizes merely widens. To establish the proportion of traffic that is limited by slow-start rather than capacity we have performed a study of Internet flow sizes based on packet trace analysis. Our initial analysis is based on a longitudinal study of CAIDA packet traces from an Internet exchange point, covering the time period 2002 – 2012. A detailed description of the used CAIDA traces was given in Deliverable 3.1.

Figure 2.2 shows the prevalence of flow sizes on the Internet, on the same log-scale as used in Figure 2.1. As can be seen in the figure, the majority of the flows on the Internet are small in size. Comparing the flow sizes between the years also suggests that the distribution of transfer sizes is not changing much as capacity grows. The prevalence of short flows is also consistent with the findings of previous studies [4, 5, 6, 7]. In relation to Figure 2.1, Figure 2.2 shows that flows large enough to make use of tens of Mb/s of capacity are rare.

2.2 Discussion

As illustrated by the analysis in Figures 2.1 and 2.2, an ever-growing proportion of a typical user's sessions is becoming limited by slow-start, not capacity. A solution for a more efficient flow-start is thus important for reducing latency over the Internet. The longer we fail to solve this flow-start problem, investing in capacity will make less and less difference to more and more people.

The IETF recently standardized an experimental RFC that allows an initial congestion window of up to 10 segments [1], as compared to the initial congestion window of 3 segments that typically follow from RFC 5681 [2]. Comparing the second and third plots in Figure 2.1 shows that increasing the initial window from 3 to 10 makes a reasonable difference for flows between 4.5kB and 15kB, but for larger flows it makes increasingly less difference. Increasing the initial window is not a scalable solution to the flow-start problem. The same argument also holds for the use of multiple parallel flows. If a source opens m parallel flows all using the same slow-start with initial window i, it only gives equivalent performance to one flow in slow-start using a larger initial window mi (but with the additional overhead of the extra flows). The total window across all flows still only doubles in the rounds after the first.

The problem with slow-start is that it always has to start from the bottom, but it gets no hard information from the network until it reaches the top. Over the years, as we make the top higher, a greater proportion of all transfers complete before they even discover that they didn't need to be cautious anyway. This is a problem of absence of information about the available capacity along the network path. Several approaches for an alternative flow-start, based on various methods to obtain information about the path, have also been proposed in the literature. Three main approaches for how to obtain information about the network path can be identified: state sharing at the sender, the use of bandwidth estimation techniques, and explicit signalling from the network.

FSE described in Section 5.2, temporal and ensemble sharing discussed in RFC 2140 [8], and the congestion manager framework described in RFC 3124 [9] all represent techniques for state sharing and joint congestion management at the sender. Although joint congestion management can allow a new flow to quickly find a suitable sending rate, it is only applicable when multiple flows share a common bottleneck. Proposals that use bandwidth estimation techniques include Swift Start [10] and the improved slow-start of RAPID [11]. Swift Start uses the packet-pair technique [12] to obtain an initial estimate of the available capacity, whereas RAPID uses a more advanced scheme based on multi-rate probe streams (or chirps). Neither of the approaches has reached real deployment. The Quick-Start extension to TCP [13, 14] is a prominent example of an approach that uses explicit signalling from the network. Quick-Start allows a desired initial sending rate to be advertised in the TCP SYN segment. Each router along the path must then agree to the desired rate or reduce the rate to an acceptable value. As it relies on modifications in both end systems and network components, Quick-Start suffers from deployment difficulties and has not seen any practical use.

2.3 Future Work within RITE

An initial analysis and quantification of the slow-start problem was outlined above. We are currently working on refining the quantification and in particular to perform some more sophisticated packet trace analysis. For the flow size analysis illustrated in Figure 2.2 we simply assume that each TCP connection consists of one single packet train, with one slow-start at the beginning and no limit to the supply of data from the application until the end. However, in practice short packet trains are probably even more prevalent than shown, because many long flows actually consist of a sequence of shorter packet trains separated by idle periods.

We are currently examining the CAIDA traces in more detail to extract the packet trains from within each IP flow. For this we use Tmix [15] to identify application data units (ADUs) within an IP flow. Given bidirectional traces of TCP connections, Tmix outputs connection vectors for each TCP connection. A connection vector models how the application at both ends of a TCP connection exchanges ADUs. Tmix measures both the size of the ADUs and durations between ADUs, which depend on the application characteristics. A sufficiently long elapsed time between ADUs within a long flow might tell about the existence of shorter packet trains.

For our future analysis we also plan to examine the correlation between flows. When multiple flows are started closely correlated in time, the packets they generate will appear as a single packet train to the network. Furthermore, the CAIDA traces we use for our longitudinal study are captured at an internet exchange point. As much of the traffic a typically user sees nowadays is served by CDNs, the data captured by the CAIDA traces may no longer be fully representative of the traffic seen by end-users. We will therefore complement our analysis with traffic traces taken where we can see all of each customer's traffic, e.g. within an ISP.

The analysis in this section illustrates the importance of the flow-start problem. It also shows that simple solutions such as increasing the initial congestion window or using multiple parallel connections are not scalable as network capacity grows. Although clearly more challenging from a deployment perspective, we believe that solutions that involve explicit signalling from the network will be required to fully solve the flow-start problem. The design of such solutions will be an important part of the continued work within RITE. While the design of suitable signalling mechanisms and their deployment considerations are part of WP2, the solutions for how to use this information in the end-systems will be an important work item within WP1.

3 Analysis of buffer latency issues for end systems

Host stacks provide buffering at both the sender and receiver, since generally applications execute outside the kernel where protocol processing is normally performed. Excessive host buffering not only adds to end-to-end latency, it also can result in excessive resource consumption. A straightforward mitigation is to limit the maximum queue size, although in practice there are many places in the stack where buffering may be required. There is evidence that operating systems are being updated to limit the effects of bufferbloat [16], therefore RITE will take care not to duplicate these efforts.

[17] showed that tuning the send socket depth to 2 x cwnd could avoid excessive buffering for bulk TCP flows without impacting throughput. TCP small queues (TSQ) also modified the Linux kernel [18] to limit the default network socket to 128KB to avoid sender latency. Moreover, [19] showed a trade-off between the depth of socket input buffering and the TCP throughput that could be used to optimise performance when an application wishes to trade the maximum throughput for reduced latency. In real-time applications (such as interactive streaming using the Real-Time Protocol, RTP), a common issue is that network latency can cause data to arrive after a deadline, and the data is then of no (or limited) value; excessive input buffering adds to this latency. While TCP's stream-based design requires data sent to the socket API to be sent on the wire, this is not a requirement for best-effort datagram-based protocols such as UDP [20] or DCCP [21]. Late-access methods allow the queued data to be updated or stale data to be removed. Alternate buffering designs are common for non-reliable transports, e.g. the use of ring-buffers and other techniques that only buffer the newest data.

Datagram-based protocols do not require significant receive buffering, leaving any re-ordering decisions to the application, where these may be kept to a minimum for latency-sensitive applications. When data is framed as separate messages (e.g. application-layer framing), it may be possible for an application to skip data that is no longer needed or avoid waiting for data that was not received. SCTP [22] permits a stream-based partial reliability in which the data can be subdivided into multiple flows each of which may be separately managed. For SCTP this matches the goal to support latency-sensitive transaction processing applications. The proposed TCP Minion API [23] replaces the upper layer interface of TCP with one that better supports the application layer. There are at present no proposals to standardise a low latency API that may be used across multiple transport protocols.

Latency can also be introduced within a protocol module when data needs to be copied between a buffer and application, kernel, or device memory. A Zero-copy technique avoids redundant copying between intermediate buffers and associated context switches [24]. This can significantly benefit both bulk transfer over high-speed networks [25] and also latency-sensitive real-time applications [26]. Both the kind of traffic and the transport protocol influence the best specific implementation of the zero-copy mechanism, though, as documented by RITE partners [27].

3.1 OS buffer delays

The presence of over-dimensioned buffers in operating systems leading to unnecessary latency has, since the writing of the proposal, been explored in depth by other projects. The adverse effects of buffering in the operating systems at different levels are now well understood, and are being handled by the research community and commercial actors. The CeroWRT router software project [28] aims to reduce buffer sizes and introduce AQM algorithms to reduce latency. The end product is deployed in a branched version of the Linux-based openWRT router software. As mentioned in the article on TCP small queues [18], the Linux community is also dealing with the problem of oversized queues. These initiatives are heavily supported by Google [29].

As other initiatives, among them the RITE collaborators in the bufferbloat project [16], are actively attacking this problem, the RITE consortium has chosen to focus its resources on other avenues of investigation where the gain of investing our efforts will be greater.

4 Analysis of loss-recovery delays

Interactions within protocols can contribute extra latency, due to non-optimised protocol design and application requirements. It is thus important for RITE to develop novel protocol-specific mechanisms that reduce latency for application-limited and short-lived flows.

A protocol that provides reliable data transfer from end-to-end can introduce a source of latency, especially when application-limited traffic such as short-lived or interactive flows are transported [30, 31]. TCP uses two loss recovery mechanisms called fast retransmit [2] and retransmission timeout [32]. Fast retransmit is generally preferred as it detects data loss much faster than retransmission timeout [33]. However, fast retransmit is often inhibited when a small amount of data is exchanged, a common situation for application-limited traffic [34, 35]. Thus, application-limited flows must rely on the much slower retransmission timeout mechanism for loss recovery.

There are several proposals in the literature that address retransmission latency issues. For TCP, the most straightforward is to relax the conditions for triggering fast retransmit, either statically for all traffic [36] or dynamically in situations where a retransmission timeout normally is the only option for loss recovery [37, 38]. Another commonly suggested approach is to change the way that data is transmitted and acknowledged, to increase the chance of using fast retransmit to detect data loss [34, 39, 40, 38]. While both approaches can be used to provide faster loss detection, they could introduce other problems. For example, network effects like packet reordering can cause unnecessary retransmissions when relaxing the conditions for fast retransmit [41]. If the amount of unnecessary retransmissions becomes too large, the performance might instead suffer. Furthermore, by modifying the way data is transmitted other

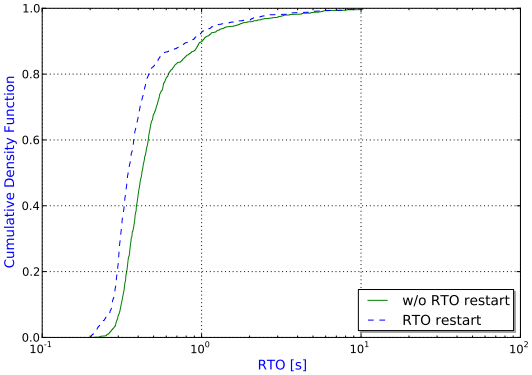

Figure 4.1: Cumulative density function showing the time required to recover from loss using RTO, with and without the RTO restart mechanism.

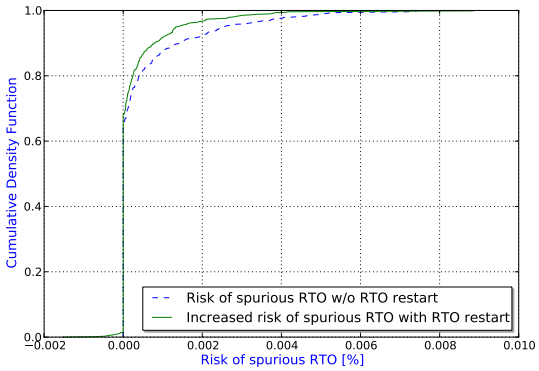

Figure 4.2: Cumulative density function showing the risk of having a spurious RTO without RTO restart and the increased risk when using RTO restart.

problems can emerge. For example, modifications need to be sufficiently compatible with standard TCP [42, 2] to be deployable in an incremental fashion.

In the coming section we report on the analysis conducted so far related to loss-recovery latency.

4.1 RTO-restart

The approach we have used thus far (RTO-restart [43]) tries to avoid the problems mentioned in the introduction of Section 4 by instead making the retransmission timeout (RTO) faster. Similarly to proposals addressing the fast retransmit mechanism, it is possible to modify the timeout mechanism by configuring it differently. This approach has been successfully demonstrated in e.g. [44, 45]. However, the problems related to such reconfiguration are similar to those when reconfiguring fast retransmit: the conservative behavior that is needed to avoid unnecessary retransmissions is relaxed, and in some situations this can lead to performance degradation instead of improvement. There are also a number of suggestions on how to change the way the retransmission timeout mechanism works, including [46, 47]. Furthermore, there are also proposals that try to avoid retransmission timeouts by using alternate timeout-mechanisms to faster detect loss [48, 49]. However, as none of these approaches have been evaluated or deployed in a sufficiently extensive manner, it is not known whether they are beneficial or sufficiently safe to use in the regular Internet.

The current RTO management algorithm in TCP [32] causes retransmissions to occur RTO[1] seconds after the last acknowledgment (ACK) has been received, not RTO seconds after the transmission of the lost segment(s). In most cases, this adds approximately one round-trip time (RTT) to the loss recovery time. If the ACK that triggers the restart also is a delayed ACK, then the total loss recovery time can become RTO + RTT + delACK, where delACK corresponds to the receiver's delayed ACK setting. This delay can be significant for time-sensitive applications.

To enable faster loss recovery for such applications we have proposed the alternative algorithm RTO-restart which is designed for situations where fast retransmit does not apply (less than four packets are outstanding). By resetting the timer to $\mathrm{RTO} - \mathrm{T}_{\mathrm{earliest}}$, where $\mathrm{T}_{\mathrm{earliest}}$ is the time elapsed since the earliest outstanding segment was transmitted, retransmissions will always occur after exactly RTO seconds. The RTO restart algorithm has been implemented in the Linux 3.7 kernel [50].

Based on our previous research related to the RTO-restart mechanism, we know that the latency reduction can be significant. Figure 4.1 shows the cumulative density function (CDF) of the expected RTO (in seconds) using the original RTO management and our RTO-restart for traffic obtained from an online game called Age of Conan. The CDF clearly shows an advantage of using RTO-restart in favor of the

[1] The retransmission timeout mechanism and the time required for timeout are both abbreviated RTO.

original mechanism.

However, RTO-restart can also increase the number of spurious timeouts, similar to the problems of spurious fast retransmits mentioned earlier. It is therefore important to quantify the probability of such events happening, to ensure that our proposed mechanism does not have an overall negative effect on the performance.

To assess this risk we have set up a physical test-bed in which we replay the Age of Conan traffic using both the standard RTO mechanism and RTO-restart. The trace file containing the Age of Conan traffic includes latency-sensitive data sent from both game clients to the server and vice versa. For our experiments we only use the data transferred from the server to the game clients. We extract RTTs, message sizes and message inter-arrival times with common tools like e.g. tcptrace and Tshark. We then emulate the gaming scenario in a three machine test-bed. One of the edge machines emulates the role of the game server and the other emulates the role of the game clients. We replay each connection from the trace as a different gaming scenario between the server and the client. The middle machine in the test-bed emulates a range of network scenarios. To construct such scenarios, the emulator delays every packet based on the RTTs seen in the original trace such that each packet transmission in the replay phase approximately the same delay as seen in the trace. Besides, all traffic between the server and the client(s) have message sizes and inter-arrival times directly taken from the real trace. Figure 4.2 shows both the risk of having a spurious RTO when using standard TCP and the increased risk when using the RTO restart mechanism. The risk, and the increased risk when using RTO restart, shown in the figure is calculated with respect to the connection size in packets. As illustrated by the figure, RTO restart does not cause any extra spurious RTOs in about 65% of the cases. Above this, there is a 0.005% increased risk of having a spurious RTO, in the worst case.

Our ongoing research related to this mechanism includes experimentation to further confirm that spurious timeouts do not make the mechanism unsuitable for general deployment. Following, and parallel to, this experimentation are our continued efforts in standardising the mechanism through the TCP Maintenance and Minor Extensions (tcpm) working group of the IETF. The current status of this standardisation process is the acceptance of our proposal as a working group item. This effectively means that the working group has agreed on collaboratively improving the specification, until it becomes a standard.

5 Analysis of capacity sharing

The Internet is a shared resource. It is therefore common that many flows share at least a portion of the network path with other flows. The impact sharing has on a specific (foreground) flow depends on the characteristics of the other (background) flows and the network capacity. We use the term *collateral damage* to indicate the effects that certain policies of congestion control have on the overall network performance in presence of shared resources. Since the congestion control behaviours of flows over the Internet are heterogeneous, it is difficult to predict how certain policies affect the performance of the traffic aggregate.

Background traffic may even use a similar congestion control algorithm to the foreground traffic, but its performance may still vary since these are in part determined by the application traffic and other parameters such as the RTT. Short-lived flows can also result in more damage, since there is less opportunity to react to congestion/delay and choose a safe sending rate [51].

Background traffic may be more responsive to congestion (e.g., LEDBAT [52]), minimising collateral damage, or it could be less responsive, including the possibility of flows that either do not react to congestion or react much more slowly than the foreground flow. The impact on the foreground flow ranges from increased jitter (short-term variation in delay) to persistent queueing - leading perhaps to additional delay, packet marking or loss (invoking additional delays).

The level of collateral damage can be reduced when background flows use appropriate transport congestion control [53] but cannot be eliminated. Burstiness of network traffic can also be mitigated at the expense of increased latency by using traffic shaping to pace packets. The impact of background flows is expected to be more when routers use tail-drop queue management procedures, but may be mitigated

using flow isolation mechanisms (such as flow classification and some form of non-FIFO queuing in the network buffers).

TCP sources injecting too large bursts of packets at line-rate into the network was identified as a potential risk of collateral damage for shared bottlenecks. new-CWV [54], a TCP modification developed in the context of RITE to provide a suitable congestion control for data-limited flows, could cause this problem when restarting idle connections. TCP congestion control design needs to consider burst mitigation techniques to avoid this kind of transient congestion.

Improving Internet congestion control using only implicit signals might not be sufficient to adequately counteract collateral damage. Flow State Exchange (FSE) is an end-to-end technique developed in RITE to provide a framework to exchange information between flows sharing a network path. The FSE should only operate on flows that traverse a common bottleneck; Shared Bottleneck Detection (SBD) techniques (presented in section 5.3) may provide a way to acquire this information via passive traffic analysis.

The following sections document the progress in RITE in engineering these techniques and briefly anticipate the work that will be done.

5.1 Congestion Window Validation (new-CWV)

This section describes the status of implementation and testing of new-CWV [55], an update to TCP standard behavior that was proposed by RITE and is being standardised in the IETF TCPM working group. We first introduce new-CWV by discussing its usage among applications that use TCP as transport. Then we describe its implementation and the type of tests so far carried out on this prototype, which highlights important design choices.

5.1.1 Applications benefitting from new-CWV

Many current Internet applications can be characterised as rate-limited. A rate-limited application is an application that transmits at a rate not directly controlled by the transport protocol, but instead dictated by the application. TCP was designed to support a range of applications, but TCP congestion control [2] has been optimised primarily for bulk transfers. In contrast to rate-limited applications, the throughput of a bulk transfer is limited by the available network capacity and TCP adjusts its congestion window in order to use as much as possible of network resources. This may induce queuing delays within bottleneck routers.

The recent growth of TCP-based multimedia and interactive web applications has reopened the debate on use of TCP for rate-limited applications [56]. A whole range of applications, which make up a significant number of Internet connections, has an evident rate-limited behaviour:

- Web browsing based on HTTP/1.1. The typical Internet web page consists of a large number of objects, which are requested and sent separately using various parallel connections. HTTP/1.1 persistency allows a connection to be reused to send multiple objects. Since the objects are typically a few kilobytes, the sequence of transmissions generates a rate-limited traffic pattern.
- A new form of interactive web based on HTTP/1.1. This category of real-time applications does not necessarily need to use all the Internet path capacity but requires conveying data to users in a prompt and timely fashion (i.e., low transfer latency).
- HTTP Adaptive Streaming (HAS) is emerging as a practical method to distribute video content over the Internet based on TCP. HAS consists of segmenting the real-time stream in equal duration chunks whose size is negotiated between sender and receiver and adjusted during the streaming depending on network congestion. HAS can be used with several encapsulation formats (RTSP, MMS, PNM, RTMP, DASH, etc); however, the traffic rate pattern generate by HAS is usually rate-limited and very bursty [57]. Such bursts have the potential to induce significant queuing in a bottleneck router, impacting the latency of other flows.

- Many other applications that use TCP, such as chat messaging, online gaming and betting, sport results, surveillance systems, sensor network monitoring, etc. can be data-limited and performance is typically latency sensitive.

The fact that standard TCP is not effective at handling rate-limited traffic motivated many application developers to use padding during application idle-periods to keep the congestion window (*cwnd*) at a large value. Although this can ensure acceptable application performance for foreground traffic, it can also degrade network performance and decrease the effectiveness of TCP congestion control [58].

Our proposal, new-CWV, introduces a set of modifications to Standard TCP to enable effective use of standards-based methods. New-CWV freezes *cwnd* during a rate-limited period, enabling the application to restart with the same *cwnd* after a rate-limited period. Hence, *cwnd* would neither grow nor shrink while rate-limited. This allows applications to more quickly resume transmission. When a new-CWV sender detects a congestion event during the first RTT of restarting with a large *cwnd*, the methods appropriately reduce *cwnd*. This can satisfy the capacity requirements of realtime applications and at the same time provide congestion control appropriate for use in the Internet.

5.1.2 Rate Limited Traffic over TCP

This section summarises the differences between the current TCP specifications (Standard TCP), the experimental RFC 2861 Congestion Window Validation (CWV), the new proposal new-CWV [55], and the default Linux behaviours for the congestion control of rate-limited applications.

Standard TCP and CWV Standard TCP dictates that when an application is idle for a period greater than the Retransmission Timeout (RTO), *cwnd* is reset to no more than the Restart Window (RW) [2]. Reducing *cwnd* to RW is done to restart from a safe value after the path has been untested for some time. However, during an application-limited period, Standard TCP allows the *cwnd* to reach an arbitrarily large value. Since packet probes are sent at a lower rate than permitted by *cwnd*, the reception of an ACK does not provide evidence that the network path is able to sustain the transmission rate reflected by *cwnd*. The result is an "invalid" *cwnd*, i.e., a poor estimate of the available path capacity. If an application with an invalid *cwnd* increases suddenly its transmission rate, the injection of a significant volume of additional traffic into the network could cause severe congestion and increase in latency for other traffic [59].

Although reducing *cwnd* to RW after an idle period is a safe action, it causes substantial performance degradation to bursty applications that require prompt restart after idle. This has been a disincentive for application designers to consider Standard TCP for real-time applications.

CWV [59] was proposed as an experimental standard in RFC 2861.

UoA simulations have shown that neither Standard TCP nor CWV are suitable for rate-limited applications [60]. Both approaches are too conservative in that they reduce *cwnd* too quickly to meet the requirements of real-time applications.

new-CWV new-CWV [55] proposes an alternative approach to address the problems of Standard TCP with rate-limited applications. The idea is to use a single approach to regulate the *cwnd* during idle and data-limited periods, a period referred to as a Non Validated Period (NVP).

To evaluate if the connection is in a NVP, a new sender variable, named *pipeACK*, is used to measure the portion of path capacity used by the connection in recent transmission history. TCP is considered in a NVP when *pipeACK* is less than $cwnd/2$. In this case, the connection consumes very little of the path capacity. Thus, new-CWV does not increase further the *cwnd* at each ACK reception. If *pipeACK* is larger than $cwnd/2$ instead, the *cwnd* can be considered "validated" and it can be grown every time an ACK is received.

The way *pipeACK* is calculated is critical in new-CWV. The next sections discuss different ways to obtain this estimate in an efficient way. New-CWV computes *pipeACK* filtering *pipeACK* samples

windows of packets for which an ACK has been received. The algorithm uses *pipeACK* samples rather than FlightSize.

The NVP concludes: (i) When the sender transmits sufficient data so that is no longer rate-limited, (ii) after 5 minutes from the beginning of the current NVP, or (iii) when a congestion signal is received.

If a sender remains in a NVP for more than the 5 minutes (case (ii)), then *cwnd* is no longer an acceptable estimate of the path capacity. new-CWV therefore reduces the *cwnd* to $max(cwnd/2, IW)$. If *ssthresh* was low compared to *cwnd*, it also increases *ssthresh* to $max(ssthresh, 3cwnd/4)$. The weighting three-fourths is to avoid excessive overshoot, as noted in [59].

Three types of congestion signals can occur during the NVP (case (ii)): An RTO expiry, the detection of packet loss, or an Explicit Congestion Notification (ECN). An RTO expiring during the NVP is considered a strong congestion signal. Hence, new-CWV terminates the NVP and uses the Standard TCP mechanism to resume from this state. Conversely, when the sender detects a packet-drop or receives an ECN mark, new-CWV follows a different approach than Standard TCP specifications.

As soon as a sender detects loss, it reduces the *cwnd* to a safe value for the network. The introduction of NVP improves the performance of an application that exhibits frequent rate-limited periods.

Linux Implementation To test new-CWV with rate-limited applications, UoA analysed the behaviour of the Linux kernel as an example TCP stack implementation.

The Linux kernel behaviour is similar to new-CWV in that it freezes *cwnd* rather than reducing it during a data-limited period. However, we suggest this behaviour is too conservative when restarting in slow start after an idle period and too aggressive when the slow start restart is disabled. Allowing TCP to keep the *cwnd* open for an arbitrarily long period of time could be potentially dangerous if the congestion state or even the path characteristics have changed significantly during this period. new-CWV proposes a NVP of a most 5 minutes. Future research may show that a smaller NVP interval could also be acceptable to applications.

5.1.3 Implementation & Testing

To establish if a connection is application-limited, we need to define a method to calculate the amount of data per RTT that TCP has transmitted and that has been validated. In new-CWV *pipeACK* is set to the largest validated FlightSize in a past period of observation, called PipeACK Sampling Period (PSP). A PSP should be sufficiently large to cover periods of bursty transmission from the application, but not too much to affect the responsiveness of congestion control.

In our implementation we assume a length of $max(3RTT, 1s)$; i.e., we consider one second sufficient to track the variations of *pipeACK*, but we consider at least three RTTs when the connection has a RTT for longer Internet paths.

We implemented new-CWV in a Loadable Kernel Module (LKM) and used the set of tcp_congestion_ops function pointers to redirect the flow of control in our module. We used a LKM as a flexible way to test multiple kernel versions and to be portable to different systems.

A key concept of new-CWV is that TCP has evidence that the path is able to sustain a certain window of data only when it receives an ACK for the last packet in the window. In our method we call *pipeACK* sample the size of a validated window. The pipeACK is the *largest pipeACK* sample in a PSP.

In our implementation we assume an approximation of *pipeACK* that uses only few *pipeACK* samples.

Figure 5.1 shows the dynamics of the *cwnd*(red line) and *pipeACK*(dark green line) after an idle period (a) and during a data-limited period (b). A light green vertical bar is depicted every time the application generates a burst. The amplitude of the bar represents the amount of data generated. new-CWV does not reduce the *cwnd* in both cases, which allows a TCP sender to send bursts as soon as data is available. *pipeACK* is able to track the envelope of the bursts and give indications about the TCP state, i.e. whether it is data-limited or network-limited.

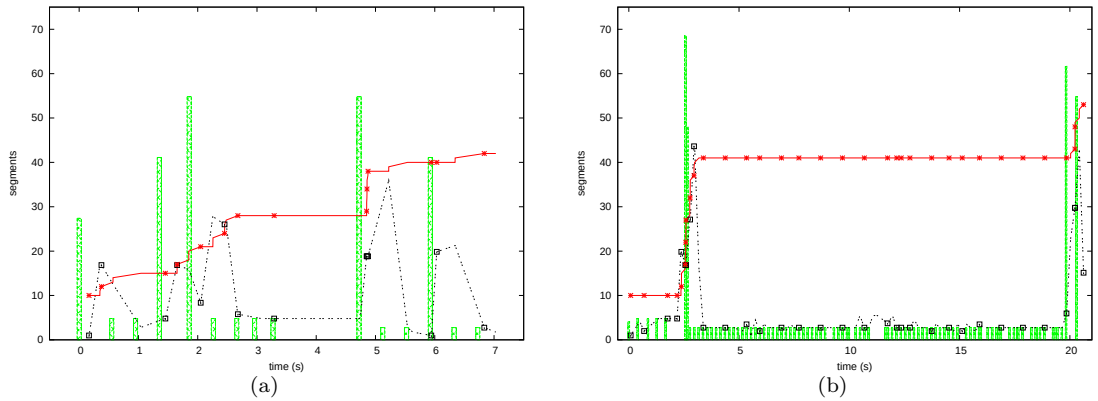

Figure 5.1: Behaviour of new-CWV during idle and application limited periods

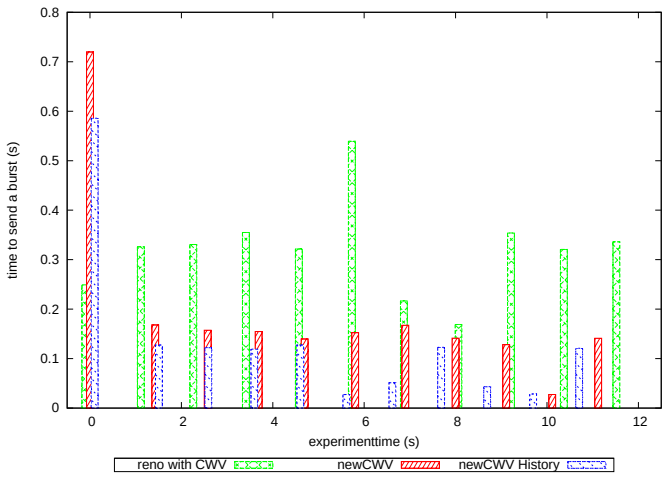

Figure 5.2: Completion times of Burst with Standard TCP, CWV and new-CWV. Bursts are 50 kB and sent every second.

Figure 5.2 shows the completion time of individual bursts of a sequence of 50 kB bursts sent every second. This graph compares the performance of the default Linux behaviour(green bars) and new-CWV with PSP duration of one RTT (red bars) and of one second (blue bars). An RTT of 200 ms was created artificially in this experiment using Netem [61]. If we exclude the first burst when *cwnd* is still increasing, new-CWV always performs better than Linux CWV. This highlights the benefits for the application when using new-CWV. *cwnd* is maintained at the same level across different bursts and does not limit the application rate (which would otherwise add latency to the responsiveness at the application level).

Figure 5.2 shows that little differences exist between a PSP=1 RTT and a PSP=1 s. Experiments showed that new-CWV is not significantly affected by this parameter. We chose a PSP of one second because this makes new-CWV slightly more responsive in increasing *cwnd*. The *pipeACK* dynamics are more stable when PSP=1 s. Since TCP does not increase the *cwnd* if *pipeACK* is less than *cwnd*/2, the *cwnd* increases less rapidly with PSP=1 RTT as an ACK has higher chances to find a low *pipeACK* when it is received.

A delayed growth of the *cwnd* can happen also during the first RTTs. new-CWV requires between one and two RTTs to estimate *pipeACK*. A minimum of one RTT is spent to receive the first ACK of a burst. The first ACK is not normally sufficient to validate the amount of data sent in the first RTT. The burst is validated only when an ACK for the last segment in it is received. Thus, another RTT might be necessary to estimate *pipeACK*. During the initial period *pipeACK* is undetermined and an initial value

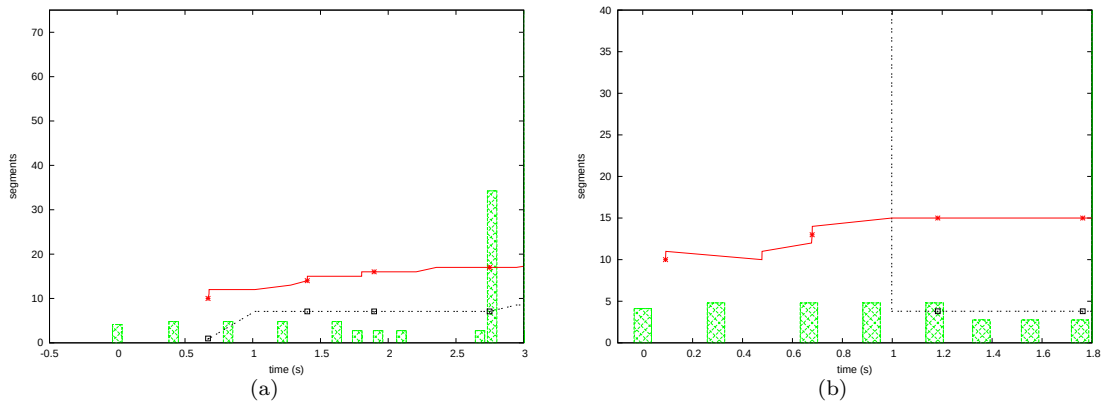

Figure 5.3: Initial *pipeACK*

instead. Setting the initial *pipeACK* to infinite avoids delaying the growth of *cwnd* during the second RTT (see Figure 5.3).

More experiments will be conducted to exercise the new protocol in a wider range of scenarios and with a richer set of applications, including web applications and real-time streaming. This will also explore the value of persistency - where a single flow can be used for multiple data transfers (and hence avoids the need for slow-start probing for each data transfer). We expect our results to demonstrate the robustness and flexibility of our method, and to persuade the Internet community to adopt this technique as a standard.

5.2 Flow State Exchange

When investigating the origins of queuing delay (as a major contributing factor to end-to-end latency), we found that it is not only affected by the *behavior* of the congestion control of flows, but also by their *number*. In simple words, two flows competing for capacity on the bottleneck tend to create more average queuing delay than a single flow that seeks out the available capacity. Hence, if there is a way to reduce the number of flows, this can play a significant role for latency reduction.

Multiple congestion controlled flows (e.g., TCP) between the same two hosts usually have separate congestion control instances, i.e. act like entirely separate flows, even when the path between them is the same. There may be several reasons for this separation. For example, one cannot always be sure if the path is indeed the same – routing mechanisms like Equal-Cost Multi-Path (ECMP) may assign different flows to different paths to achieve load balancing, even when they have the same destination IP address.

Some routers or other middle-boxes might identify flows using a five-tuple of source and destination IP address, transport protocol, and the transport protocol's port numbers. When – as envisioned within the emerging RTCWEB standard for interactive communication between web browsers – multiple flows are multiplexed over a single UDP port pair, they are most certainly regarded as a single flow inside the network and therefore treated in the same way. This means that it might be possible to readily apply congestion management to RTCWEB.

The new "RTP Media Congestion Avoidance Techniques" (RMCAT) IETF Working Group intends to develop standards for RTP-based interactive real-time media. RTCWEB being the major use case for these standards, RMCAT will also standardize methods for coupled congestion control, with the goal of reducing the overall delay and having the best possible control over the send rate allocation. Here, we describe RITE's proposal for RMCAT's coupled congestion control. RMCAT's congestion control should be applicable but not limited to RTCWEB. This means that we may need to jointly control flows that reside within a single application (a web browser, in case of RTCWEB) or in multiple applications. In the latter case, this may lose RTCWEB's possible benefit of knowing that packets from multiple flows will be routed in the same. There are, however, measurement-based methods to determine whether multiple

flows share a bottleneck in the network (cf. [62] and [63] for prior related work by RITE members). Being able to make use of such measurements when necessary, and supporting various intra- as well as inter-application scenarios calls for a congestion management architecture that is much simpler than, e.g., the well-known but little deployed Congestion Manager (CM) [64].

We have opted for an approach that minimizes the amount of necessary changes to existing applications. It involves a central storage element called "Flow State Exchange" (FSE). An FSE is passive in that it does not take control in deciding the rates of participating flows – in our model, flows store information in the FSE, and they poll the FSE for their next send rate. Groups of flows which should be controlled together have a common "Flow Group Identifier" (FGI), which can be set by a "Shared Bottleneck Detection" (SBD) module based on measurements or knowledge about multiplexing (as with RTCWEB). An SBD module can be a part of one of the applications using the FSE, or it can be a standalone entity; we will describe its design in section 5.3.

For each flow in a group, the FSE stores the most recently calculated rate, the desired rate (infinity in case of greedy flows), and a flow number. By definition, flow 1 has been active for the longest time and therefore has the best knowledge about the state of the network path. If all flows in a group are greedy, the actual send rate for each flow in the group is therefore determined by dividing the first flow's calculated rate by N, the number of flows per FGI (although other policies, e.g., to stop relying on flow 1 if that flow is application-limited could also be implemented).

Dividing by N gives each flow the same share of the available capacity; changing the fairness policy is a simple matter of changing this function. If a flow's desired rate indicates that it does not have enough data to send, the calculation can be adjusted to give the remaining capacity to other flows. When a group's flow 1 leaves, that group's flow 2 becomes flow 1. It can then immediately take over because the FSE always contains every flow's calculated rate. We have carried out early tests with a standalone implementation of the FSE, and two instances of the open source video conferencing tool "vic"[2] that we extended to talk to the FSE using Unix domain sockets. The vic variant that we use includes TFRC [65] congestion control (implemented and tested by Soo-Hyun Choi for his Ph.D. thesis [66]). In our changed version, vic stores the TFRC-calculated rate in the FSE and then determines its actual rate via the FSE as explained above. Our experiments are with a single physical host, using an instance of VirtualBox with Linux for the sender, running two instances of vic for the senders as well as an FSE, and an instance of VirtualBox with Linux for the receiver, running two instances of vic for the receivers. The two VirtualBox instances are logically interconnected on our Mac OS X host system, and the outgoing interface of the sender is set to have a maximum rate of 1 Mbit/s and introduce a propagation delay of 50 ms using tc / netem.

This section presents a preliminary result from a test where we have configured the FSE to let the two flows share the capacity equally, similar to what TFRC would automatically converge to. This helps to highlight the impact of merely using an FSE without manually influencing the fairness between flows. To make the test repeatable, we have made vic play a file (the common "foreman" test sequence), causing it to adjust the frame rate, which translates into the received video slowing down in the face of congestion. Figure 5.4 shows the sending rates of the two sender-side vic processes without the FSE, and Figure 5.5 shows their sending rates with the FSE.

The test ran for two minutes. Process 1 transmitted data from the start, whereas process 2 was started after 30 seconds and left to run for one minute. Clearly, Figure 5.5 shows less rate fluctuations than Figure 5.4 in the period when process 2 was active. Process 2 needed no start-up phase with the FSE – it directly jumped to the correct rate, determined by process 1. In doing so, it also did not exceed the bottleneck capacity, which created delay and eventually caused packet loss without the FSE; on average, the delay measured with ping throughout the test was 19% higher in the case without the FSE.

The FSE algorithm, and evaluations in ns-2, are work in progress; the most recent version of the algorithm is specified in [67]. We are planning to investigate the trade-offs of implementing the FSE algorithm *actively* (where all flows have to immediately use the newly calculated rates) vs. *passively* (where each flow only updates its own rate when its congestion controller has derived a rate update). We will study more conservative variants of the algorithm that may perform better regarding our goals of reducing

[2] http://mediatools.cs.ucl.ac.uk/nets/mmedia/

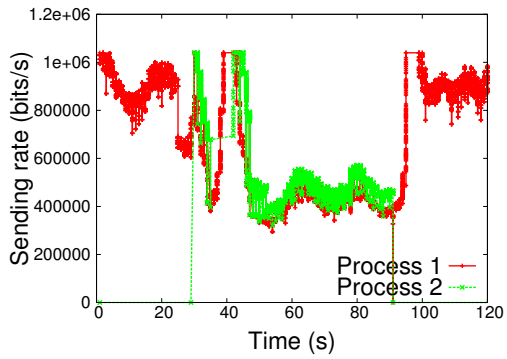

Figure 5.4: Sending rates of two separate vic processes using TFRC across a 1 Mbit/s, 50 ms link.

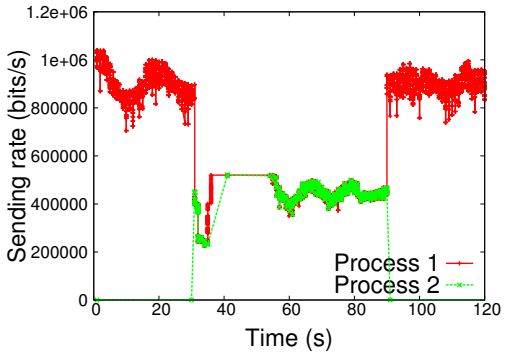

Figure 5.5: Sending rates of two vic processes across a 1 Mbit/s, 50 ms link when TFRC congestion controls are coupled via the FSE.

queue growth and packet loss, and we will then turn towards window-based controls too, starting with LEDBAT, and eventually looking at TCP.

5.3 Shared bottleneck detection

Shared bottleneck detection is an enabling technology for latency reducing endeavours that deal with multiple flows on multiple paths, such as FSE and multipath transport protocols. Most of the work in the literature on shared bottleneck detection is based on active probing techniques ([63] gives a good overview). Our work here is passive, relying only on the packets normally sent by each flow. The two main base metrics available are: packet loss and packet delay. Both of these metrics have their problems: If the bottleneck has a large buffer, there can be a long time interval between loss events, making correlation based on this slow and work only when network latency is high. Also, if a flow's path includes a lossy link, such as a wireless link which is not a bottleneck, loss on this link will dominate the statistics masking the actual bottleneck's loss signal. The delay signal tends to be very noisy. A bottleneck link will induce relatively higher delays to packets, but not every packet will experience the high delays. Also, generally uncongested links can experience momentary congestion adding to the noise on the end-to-end delay signal.

Summarising previous work by RITE members (more related work will be covered in a paper that is currently under preparation): Yousaf et al. [62] calculates a full cross-correlation of the One-Way-Delay (OWD) signals, filtering the noise with Singular Value Decomposition (SVD). This can achieve good results in some circumstances, but is calculation intensive. Hassayoun et al. [63] propose a heuristic based on congestion event signals (initiated with loss, but also using delay) occurring at similar times. The mechanism is designed to work in combination with multipath congestion control.

In an effort to achieve better precision in a practical system, improving upon the related work, we seek to develop a robust mechanism with the following properties:

- Independent of any particular congestion control mechanism or application (thus suitable for use in a FSE).
- Accurate and timely detection and grouping.
- Work with both large and small buffers.

To date we have investigated two promising OWD based methods. Our first method is based on the observation that bottlenecks introduce wide variations in packet delay as they become congested. Packet Delay Variation (PDV) is a performance measure defined in RFC 5481[68], ITU-T Y.1540[69] and Y.1541[70]

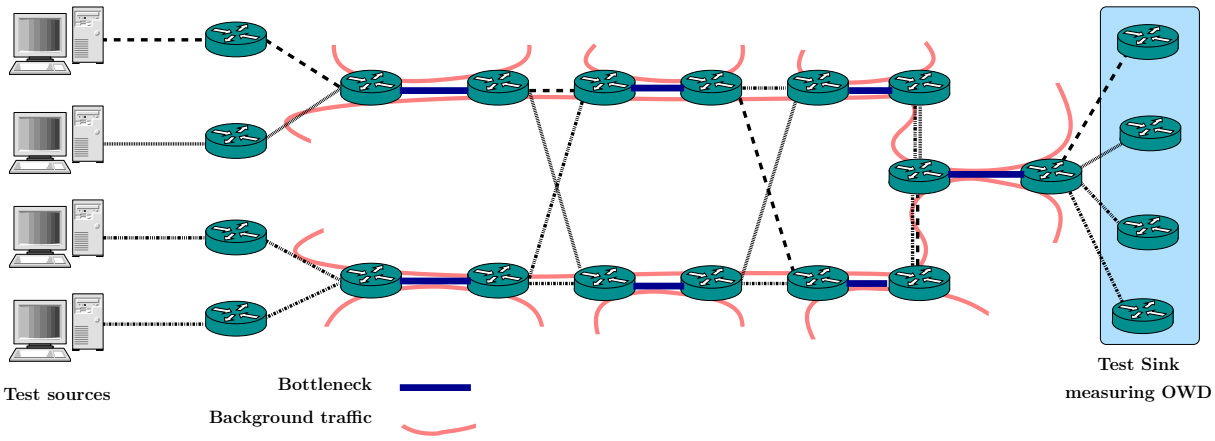

Figure 5.6: Shared bottleneck detection NS2 simulation set up

as $\text{PDV}_1 = d_{\max} - d_{\min}$ or $\text{PDV}_2 = \bar{d} - d_{\min}$ in the measurement period It assumes that flows with a common bottleneck will also have a similar end-to-end PDV. Our method groups flows with a similar PDV together. We have found that PDV_1 accurately groups flows sharing a bottleneck when all flows send enough packets for an adequate sample size in the measurement period, and all flows send a similar number of packets. When flows send at different rates, we find that PDV_1 is prone to grouping errors due to different estimates of $d_{\max}$ or $d_{\min}$, these two being the tails of the distribution. PDV_2 is able to group flows better in these circumstances, using a $d_{\min}$ calculated over a longer period than the measurement period.

We attempt to group only flows that traverse common congested links. It is difficult to determine the congested state of a path using PDV alone as both lightly congested and extremely heavily congested bottlenecks will have a small PDV. An additional measure is needed for this, and is the subject of further investigation. Another concern is that PDV may not be able to provide a large enough discrimination for shared bottleneck detection if buffers are small, a key objective for reducing latency in the Internet.

The second method we are considering is to use the shape of the OWD distribution as a measure of congestion and to group flows sharing a common bottleneck. We postulate that OWD shape is different for congested and uncongested paths, and that it will be similar for flows that traverse the same bottleneck. Preliminary ns2 simulation experiments confirm this.

Figure 5.6 shows the simulation setup used for the initial tests. Most of the traffic is realistic background traffic generated by Tmix using the TCP evaluation suit traces (available at `http://hosting.riteproject.eu/tcpevaltmixtraces.tgz`). The test sources send packets at different rates to the test sinks, where the OWD is measured. Bottlenecks are created by reducing the capacity of the targeted link. The setup allows techniques to be tested for different combinations of pairs of flows sharing one or more bottlenecks.

A simulation environment allows us to test the mechanism's operation in scenarios which we envisage to be more difficult than normal network conditions, and where the ground truth concerning bottlenecks and correct groupings is known. After proving the technique robust in simulation, we plan to test it on the NorNet testbed (`http://www.nornet-testbed.no/`) using a combination of wired and wireless (3G) nodes. The NorNet testbed is built over the actual Internet and 3G network.

In a real network the ground truth concerning bottlenecks during the tests cannot be determined. We plan to verify bottleneck locations and necessary traffic loads using probing techniques. Based on this, we plan to generate realistic traffic from various locations in the network to create bottlenecks and test our mechanism. This will give us confidence in the mechanism's operation in a real network and insight into the signal it can provide for latency reducing mechanisms that work with multiple flows on multiple paths.

6 Analysis of API latency issues

Transport protocols such as SCTP [71] or the recently proposed "TCP Minion" [23] allow for latency reduction via transport service differentiation. For example, if an application can cope with data arriving out-of-order, such an application can get data transmitted faster with SCTP or Minion than with TCP, because TCP enforces in-order delivery at the expense of some delay. Such communication, however, requires the application programmer to be in the loop. Nowadays, an explicit choice for e.g. SCTP must be made, rather than simply specifying that out-of-order delivery is desired. This introduces a significant burden:

- SCTP (in kernel space) is not available on all Operating Systems. When it is available on one host, it may not be available on the other end, or not work across the Internet path. An application must therefore check if SCTP communication works, and otherwise fall back to a less efficient means of communication (typically TCP). Such a fall-back, though possible (see e.g. [72]), is not easy to implement efficiently.
- There may be other opportunities: SCTP could be run over UDP, which incurs more overhead and may in general be less favourable than "native" SCTP if such an SCTP implementation exists and if packets pass through the network (e.g., there are issues with NATs and SCTP). Another alternative is to embed a user-space implementation of SCTP-over-UDP in the application, but this may be less efficient than using kernel-space code.
- Alternatives such as TCP Minion could be used. This requires at least the other side to be aware of what is going on, and availability on both sides must therefore be checked.

As another latency-related shortcoming of today's transport API, RFC 6897 [73] identifies low latency and jitter stability as one out of (at least) three performance objectives for multipath transport. This RFC goes on to state, as a requirement for an advanced future API:

> An application should be able to inform the MPTCP implementation about its high-level performance requirements, e.g., in the form of a profile.

6.1 Service specific communication with OS

With the current transport API, the amount of work involved in trying to use one of the mechanisms for low latency is exceedingly large. It would therefore be much better if this whole functionality could be hidden underneath an API that allows an application to simply specify a service instead of a protocol.

Making this possible requires a larger organizational effort. To this end, RITE members from IMT and UiO have teamed up with members of the Trilogy 2 project to work towards a "Birds-of-a-Feather" (BOF) session at the 89th IETF meeting in London in March 2014, with the goal of forming an IETF Working Group that would standardise transport services. Up-to-date information on this work along with earlier background material is available at `https://sites.google.com/site/transportprotocolservices/`. An Internet Draft containing the problem statement has already been made available to the IETF community [74], and it will be presented at the 88th IETF meeting in Vancouver, in November 2013.

7 Analysis of multipath transport protocols in regard to latency

Increasing link speeds would seem like an obvious way of both reducing packet serialization times, and alleviating persistent congestion that yields large queuing delays[3]. However, such a brute-force approach

[3] Higher link transmission rates may not necessarily yield lower latency, even if both serialization and queuing delays get shorter. This can be due to e.g. TCP's behavior in slow start. However, we do not focus on such issues here; they are discussed in § 2.

to bring down latency due to capacity constraints is not always economically sensible, nor technically feasible.

An alternate way of getting more available capacity is to use several links at once. Nowadays, many end-hosts are equipped with multiple network interfaces. Transport protocols implementing *multipath* capabilities are able then to exploit multihoming in end-hosts. When using a multipath transport protocol, traffic is stripped across several interfaces simultaneously. Such is the approach followed by protocols like Multipath TCP (MPTCP) [75, 76, 77] and Concurrent Multipath Transfer for SCTP (CMT-SCTP) [78], as well as by a previous proposal by RITE members [63] that may apply to both MPTCP and CMT-SCTP. One of the main goals of these mechanisms is to maximise throughput by regarding all available links as a single, pooled resource.

Using the aggregate capacity of several links at once could in principle result in shorter flow completion times. However, in many cases such improvement may happen only with relatively large flows. Chen et al. [79] report smaller median transfer times when downloading moderately-sized (512 KB) files using MPTCP over both an LTE and a WiFi link, as compared to using a single TCP flow over the same WiFi link only. However, their results for smaller flows show little or no traffic being sent via the LTE link, so MPTCP offers no improvement with respect to TCP. Protocol overhead, related to the setting up of subflows across paths other than the primary one, is one of the reasons why use of multiple paths may not yield lower latency [80]. Such overhead means that the added latency may not make it worthwhile to actually use more than one interface, if the flow to be transferred is very short. Large differences in RTTs add to this effect, if e.g. the first subflow goes along the shortest RTT path, so the transfer may be over even before the handshake on the longest RTT path has finished [79]. RFC 6824 [80] suggests a heuristic for deciding when to start new subflows in an MPTCP connection, however, further research is still needed to assess this and other possible heuristics.

Another latency-related issue with multipath transports is that of packet reordering at the receiver [81]. Use of several paths may result in packets arriving out of order, and large differences in path RTTs may increase the severity of reordering events. Wide differences in loss rates across paths can also have a strong impact on transfer times; use of a *single* path with a large RTT but a small loss rate may perform as well, if not better, than use of such path in combination with another one, having a smaller RTT but a larger loss rate [79].

Scheduling for concurrent multipath transmission has been the focus of several previous studies, such as [82, 83, 84, 85, 86]. Adequate scheduling could help in avoiding reordering, while at the same time minimising delivery times to the receiver, but such studies tend to focus mostly on greedy flows issuing from bulk data applications, and/or on throughput optimisation.

The multipath protocols mentioned first have been designed mostly with throughput and resilience in mind, with lower transfer times as a side effect, whereas other works explicitly aim for low delay as well. Proposals such as [87] and [88] combine the use of forward-error correction (FEC) with simultaneous transmission across multiple paths; [88] is intended for UDP-based flows, whereas the design of [87] includes per-path, TCP-like congestion control. Further, MPLOT [87] considers the tradeoff between FEC and delay by adapting the amount of redundancy as a function of per-path RTTs. In both proposals, using a specific scheduling policy for striping packets (both data and FEC) across paths is key to attaining lower latency.

7.1 Exploiting multipath communication for achieving lower latency

As shown by the analysis of previous research, the use of multipath communication may not necessarily be beneficial for latency, and the majority of works in the field have so far concentrated on throughput maximisation. How to best leverage multiple paths for improving latency is still an open issue we plan to address during the remainder of the project, mainly centering on the following two topics:

- To the best of our knowledge, multipath scheduling for improving the latency of application-limited flows is still a little-covered topic. We plan on further exploring this. Heterogeneity in path characteristics and packet reordering issues will be the two main factors considered in our studies.

Task objectives	Subprojects	Maturity level	Depends on	Feeds task
OS buffer delays	N/A	Analysed. Not to be continued	None	None
Loss-recovery delays	RTO-restart	Analysed, Preliminary results	Improved methods for sharing capacity can reduce loss	1.2, 1.4 2.3, 2.4
Sharing capacity	New-CWV, Flow State Exchange, Shared Bottleneck detection, Scaling Transport Dynamics	Analysed, Preliminary results	Can be improved by network/end system interaction techniques (deliverable 2.1)	1.2, 1.3, 1.4, 2.3, 2.4
API latency issues	Service specific communication with OS	Analysed, preliminary work started	Partly dependent on loss-recovery delays, capacity sharing and multipath latency issues	1.3, 1.4, 2.3, 2.4
Multipath latency issues	Scheduling for low-latency multipath communication, Combining multipath with redundancy	Analysed, preliminary work planned to start in early 2014	Partly dependent on protocol-specific delays and capacity sharing	1.2, 1.3, 1.4, 2.3, 2.4

Table 8.1: Table summarising the status of each of the main topics for Task 1.1 and their dependencies.

- More straightforward approaches than [87, 88] mixing redundancy (for faster loss recovery) with multipath transmission could be envisioned. For instance, Vulimiri et al. [89] make the case for simply replicating packets over several paths, for low-rate, time-sensitive flows; their preliminary experiments show improvements in both the tail of the latency distribution and the loss rate. More work in this area is needed to assess the practical feasibility of simple redundancy techniques such as this.

8 Summary of analysis

The analysis phase of RITE has covered a broad range of different approaches to the problem of Internet transport latency. This section shows how the different elements of analysis and preliminary experiments complement each other and provides a firm foundation for the planned future tasks of the project. We also view the research to this point in light of the objectives for work package 1.

The bulk of our preliminary work has been on capacity sharing and loss-recovery delays since the early maturity of these topics will increase the impact of other upcoming research activities into which they feed, like APIs and multipath. For a summary of the status of the main topics and their dependencies, see table 8.1. Making modifications to transport protocols are a basic building block that we need to establish transport systems that will help reduce latency. The cooperation- and path analysis tools of *flow state exchange* and *shared bottleneck detection* will help the protocols make more intelligent decisions. Finally, an API for smarter and easier choice of transport services can be fueled by the other building blocks, providing easy to use, low-latency transport classes. The end-host mechanisms must then be considered in conjunction with the activities in work package 2 to make sure the end-hosts and network interact to provide the best possible latency.

The analysis phase has also resulted in a survey paper of latency-reducing techniques that will be submitted to a fitting venue in the Fall of 2013. A two-page position paper summarising the essence of this survey can be found on the webpages of the ISOC/RITE latency workshop that was held in September

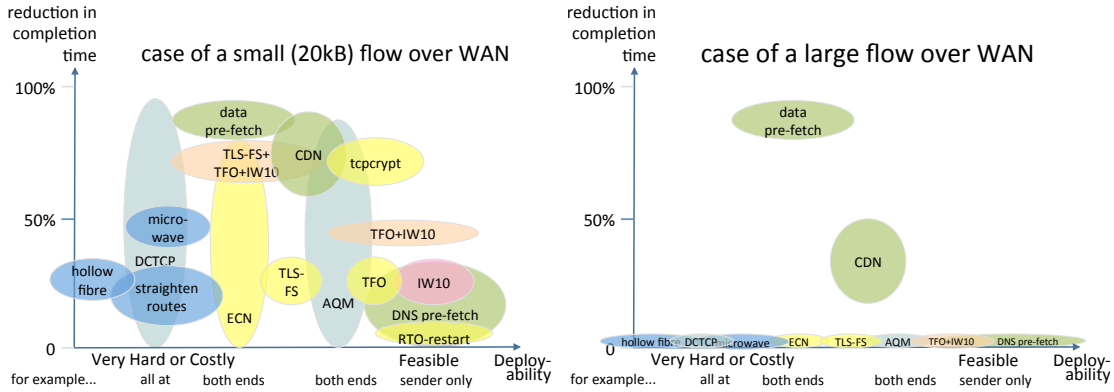

Figure 8.1: Examples of tradeoffs between deployability and latency reduction for several techniques (taken from [90]).

Objective	Subprojects addressing objective
Get a deep understanding of the latency/throughput tradeoff	All subprojects
Make TCP more suitable to low-latency applications	RTO-restart, new-CWV, Flow state exchange, Shared bottleneck detection, Scaling transport dynamics, Multipath initiatives
Make non-TCP protocols more suitable to achieve low latency	RTO-restart, Flow state exchange, Shared bottleneck detection, Scaling transport dynamics, Multipath initiatives
Make TCP less aggressive against competing low-latency apps (low latency/bulk interactions)	new-CWV, Flow state exchange, Shared bottleneck detection
Better exploiting interface diversity to achieve low latency and resilience	Service-specific communication with OS, Multipath initiatives
Optimise multiple-session resource sharing for low latency	Flow state exchange, Shared bottleneck detection, Service-specific communication with OS, Multipath initiatives

Table 8.2: The objectives of Work Package 1 and subprojects addressing the objectives.

2013[4]. The latency survey aims not only to list the state-of the art in Internet latency reduction techniques, but also to quantify the possible latency gain by applying different kinds of techniques. Using this analysis as a tool has been of great value to the consortium when prioritising between topics and for choosing what topics to focus on. In particular, one important goal of the analysis and survey work is to highlight tradeoffs between gains in latency reduction and ease of deployment, as illustrated in Figure 8.1.

Table 8.2 list the global objectives of Work Package 1 and the subprojects addressing each objective. The overall analysis has, from different angles, shed light on the tradeoffs of reducing transport latency. We have analysed and explored several options for making TCP and other protocols more suited to achieve low latency. Sharing behaviour and interface diversity has also been examined from several different viewpoints. The analysis and preliminary work will, from this point, be developed into more mature prototypes to be evaluated in prototype setups and, finally, in RITE testbed deployments (see deliverable 3.1).

Through the analysis and the preliminary work, we have extended the partners' knowledge about which avenues of investigation that is more likely to bring a latency reduction and how to approach the core RITE problems. The various skillsets of different partners are woking as a catalyst for this process, resulting in a multi-faceted view of the problems to be solved.

Table 8.3 gives a list of which RITE partner is active in the subprojects described in this document. All

[4] http://www.internetsociety.org/latency2013

RITE subproject	Collaborating partners
RTO-restart (§ 4.1)	KaU, SRL, UiO
New-CWV (§ 5.1)	UoA, SRL
Flow State Exchange (§ 5.2)	UiO
Scaling transport dynamics (§ 2)	KaU, BT, SRL
Shared bottleneck detection (§ 5.3)	UiO
Service-specific communication with OS (§ 6)	UiO, UoA
Multipath initiatives (§ 7)	IMT, SRL, KaU

Table 8.3: RITE subprojects in Work Package 1 and the partners collaborating on the respective subproject.

partners have collaborated on the latency analysis and on identifying the most promising topics based on this analysis. The partners listed in table 8.3 are the partners who have been involved in the activities during the reporting period. There will probably be other constellations of involved partners as work in the project progresses and as more subprojects move into a experimental/testbed stage.

9 Conclusions

This document has summarised the results of the analysis phase of work package 1 in the RITE project. We have presented analysis of the core topics of the project presenting the most important conclusions drawn from it. We have described the initial work done for the most mature topics, described the planned work for the topics in a less mature stage and explained why one topic has been discarded from further investigation.

The analysis phase has provided the needed insight to make the best choices between the possible future paths of investigation in the RITE project. The subtopics chosen for continued exploration are promising and provide the needed basis for the coming tasks according to the Description of Work.

References

[1] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis, "Increasing TCP's Initial Window," RFC 6928 (Experimental), Internet Engineering Task Force, Apr. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6928.txt

[2] M. Allman, V. Paxson, and E. Blanton, "TCP congestion control," RFC 5681 (Draft Standard), Internet Engineering Task Force, September 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5681.txt

[3] CAIDA, "University of california, san diego supercomputer." [Online]. Available: http://www.caida.org

[4] F. Qian, A. Gerber, Z. M. Mao, S. Sen, O. Spatscheck, and W. Willinger, "TCP revisited: a fresh look at TCP in the wild," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference, IMC '09*. Chicago, Illinois, USA: ACM, November 2009, pp. 76–89.

[5] N. Brownlee and K. C. Claffy, "Internet stream size distributions," in *Proceedings of ACM SIGMETRICS*, Marina Del Rey, California, USA, June 2002, pp. 282–283.

[6] W. Fang and L. Peterson, "Inter-AS traffic patterns and their implications," in *Proceedings of IEEE GLOBECOM*, vol. 3, Rio de Janeiro, Brazil, 1999, pp. 1859–1868.

[7] K. Thompson, G. J. Miller, and R. Wilder, "Wide-area Internet traffic patterns and characteristics," *IEEE Network*, vol. 11, no. 6, pp. 10–23, 1997.

[8] J. Touch, "TCP Control Block Interdependence," RFC 2140 (Informational), Internet Engineering Task Force, Apr. 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2140.txt

[9] H. Balakrishnan and S. Seshan, "The Congestion Manager," RFC 3124 (Proposed Standard), Internet Engineering Task Force, Jun. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3124.txt

[10] C. Partridge, D. Rockwell, M. Allman, R. Krishnan, and J. Sterbenz, "A swifter start for TCP," BBN, Tech. Rep. 8339, Mar. 2002.

[11] V. Konda and J. Kaur, "RAPID: Shrinking the congestion-control timescale," in *Proceedings of IEEE INFOCOM*, Rio de Janeiro, Apr. 2009.

[12] S. Keshav, "A control-theoretic approach to flow control," in *Proceedings of ACM SIGCOMM*. New York, NY, USA: ACM, 1991, pp. 3–15. [Online]. Available: http://doi.acm.org/10.1145/115992.115995

[13] S. Floyd, M. Allman, A. Jain, and P. Sarolahti, "Quick-Start for TCP and IP," RFC 4782 (Experimental), Internet Engineering Task Force, Jan. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4782.txt

[14] P. Sarolahti, M. Allman, and S. Floyd, "Determining an appropriate sending rate over an underutilized network path," *Computer Networks*, vol. 51, no. 7, pp. 1815–1832, May 2007.

[15] M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, and F. D. Smith, "Tmix: a tool for generating realistic TCP application workloads in ns-2," *ACM SIGCOMM Computer Communications Review*, vol. 36, no. 3, pp. 65–76, July 2006.

[16] "The Bufferbloat projects," Oct. 2013. [Online]. Available: http://www.bufferbloat.net/

[17] J. Semke, J. Mahdavi, and M. Mathis, "Automatic TCP buffer tuning," *ACM SIGCOMM Computer Communications Review*, vol. 28, no. 4, pp. 315–323, Oct. 1998. [Online]. Available: http://doi.acm.org/10.1145/285243.285292

[18] J. Corbet, "TCP Small Queues," Linux Weekly News, Jul. 2012. [Online]. Available: https://lwn.net/Articles/507065/

[19] A. Goel, L. Abeni, C. Krasic, J. Snow, and J. Walpole, "Supporting time-sensitive applications on a commodity OS," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 165–180, Dec. 2002. [Online]. Available: http://doi.acm.org/10.1145/844128.844144

[20] J. Postel, "User datagram protocol," Internet Engineering Task Force, RFC 768, August 1980. [Online]. Available: http://www.rfc-editor.org/rfc/rfc768.txt

[21] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," RFC 4340 (Proposed Standard), Internet Engineering Task Force, Mar. 2006, updated by RFCs 5595, 5596, 6335, 6773. [Online]. Available: http://www.ietf.org/rfc/rfc4340.txt

[22] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, and P. Conrad, "Stream Control Transmission Protocol (SCTP) partial reliability extension," RFC 3758 (Proposed Standard), Internet Engineering Task Force, May 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3758.txt

[23] M. F. Nowlan, N. Tiwari, J. Iyengar, S. O. Aminy, and B. Ford, "Fitting square pegs through round pipes: unordered delivery wire-compatible with TCP and TLS," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 28–28. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228337

[24] B. P. Swenson and G. F. Riley, "A New Approach to Zero-Copy Message Passing with Reversible Memory Allocation in Multi-core Architectures," in *Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation*, ser. PADS '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 44–52. [Online]. Available: http://dx.doi.org/10.1109/PADS.2012.3

[25] T. Suzumura, M. Tatsubori, S. Trent, A. Tozawa, and T. Onodera, "Highly scalable web applications with zero-copy data transfer," in *Proceedings of the 18th international conference on World wide web*, ser. WWW '09. New York, NY, USA: ACM, 2009, pp. 921–930. [Online]. Available: http://doi.acm.org/10.1145/1526709.1526833

[26] E. Kohler, M. Handley, and S. Floyd, "Designing DCCP: Congestion control without reliability," *ACM SIGCOMM Computer Communications Review*, vol. 36, no. 4, pp. 27–38, Aug. 2006. [Online]. Available: http://doi.acm.org/10.1145/1151659.1159918

[27] P. Halvorsen, T. A. Dalseng, and C. Griwodz, "Assessment of linux' data path implementations for download and streaming," *International Journal of Software Engineering and Knowledge Engineering*, vol. 17, no. 4, pp. 465–481, 2007. [Online]. Available: http://dx.doi.org/10.1142/S0218194007003343

[28] "The CeroWRT Bufferbloat project," Oct. 2013. [Online]. Available: http://www.bufferbloat.net/projects/cerowrt

[29] "Patch implementing TCP small queues (TSQ)," Linux Kernel Mailing List, Oct. 2013. [Online]. Available: http://lwn.net/Articles/506237/

[30] K. Chen, C. Huang, and C. Lei, "An empirical evaluation of TCP performance in online games," in *Proceedings of ACM ACE*, Hollywood, USA, June 2006.

[31] C. Griwodz and P. Halvorsen, "The Fun of using TCP for an MMORPG," in *Proceedings of the 2006 International Workshop on Network and Operating Systems Support for Audio and Video (NOSSDAV)*, New York (USA), 2006.

[32] V. Paxson, M. Allman, J. Chu, and M. Sargent, "Computing TCP's retransmission timer," RFC 6298 (Proposed Standard), Internet Engineering Task Force, June 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6298.txt

[33] S. Rewaskar, J. Kaur, and F. Smith, "A performance study of loss detection/recovery in real-world TCP implementations," in *Proceedings of IEEE ICNP*, Beijing, China, October 2007.

[34] H. Balakrishnan, "Challenges to reliable data transport over heterogeneous wireless networks," Ph.D. dissertation, University of California at Berkeley, Berkeley, USA, August 1998.

[35] P. Hurtig, "Fast retransmit inhibitions for TCP," Master's thesis, Karlstad University, Karlstad, Sweden, February 2006.

[36] A. Petlund, K. Evensen, C. Griwodz, and P. Halvorsen, "TCP mechanisms for improving the user experience for time-dependent thin-stream applications," in *Proceedings of IEEE LCN*, Montreal, Canada, October 2008.

[37] M. Allman, K. Avrachenkov, U. Ayesta, J. Blanton, and P. Hurtig, "Early retransmit for TCP and Stream Control Transmission Protocol (SCTP)," RFC 5827 (Experimental), Internet Engineering Task Force, April 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5827.txt

[38] D. Ciullo, M. Mellia, and M. Meo, "Two schemes to reduce latency in short lived TCP flows," *IEEE Communications Letters*, vol. 13, pp. 806–808, October 2009.

[39] M. Allman, H. Balakrishnan, and S. Floyd, "Enhancing TCP's loss recovery using limited transmit," RFC 3042 (Proposed Standard), Internet Engineering Task Force, January 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3042.txt

[40] M. Mellia, M. Meo, and C. Casetti, "TCP smart framing: A segmentation algorithm to reduce TCP latency," *IEEE/ACM Transactions on Networking*, vol. 13, pp. 316–329, April 2005.

[41] N. Dukkipati, M. Mathis, Y. Cheng, and M. Ghobadi, "Proportional rate reduction for TCP," in *Proceedings of the 11th ACM Internet Measurement Conference (IMC'11)*, Berlin, Germany, November 2011.

[42] J. Postel, "Transmission control protocol," RFC 793 (Internet Standard), Internet Engineering Task Force, September 1981. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[43] P. Hurtig, A. Brunstrom, A. Petlund, and M. Welzl, "TCP and SCTP RTO restart," Internet Draft draft-ietf-tcpm-rtorestart, work in progress, September 2013. [Online]. Available: http://tools.ietf.org/html/draft-ietf-tcpm-rtorestart

[44] M. Allman and V. Paxson, "On estimating end-to-end network path properties," in *Proceedings of ACM SIGCOMM*, Cambridge, USA, August 1999.

[45] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. Andersen, G. Ganger, G. Gibson, and B. Mueller, "Safe and effective fine-grained TCP retransmissions for datacenter communication," in *Proceedings of ACM SIGCOMM*, Barcelona, Spain, August 2009.

[46] H. Ekstroem and R. Ludwig, "The peak-hopper: A new end-to-end retransmission timer for reliable unicast transport," in *Proceedings of IEEE INFOCOM*, Hong Kong, March 2004.

[47] A. Kesselman and Y. Mansour, "Optimizing TCP retransmission timeout," in *Proceedings of the 4th Internation Conference on Networking (ICN)*, Reunion Island, France, April 2005.

[48] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan, "Reducing web latency: the virtue of gentle aggression," in *Proceedings of ACM SIGCOMM*, Hong Kong, China, August 2013.

[49] N. Dukkipati, N. Cardwell, Y. Cheng, and M. Mathis, "Tail Loss Probe (TLP): An algorithm for fast recovery of tail losses," Internet Draft draft-dukkipati-tcpm-tcp-loss-probe, work in progress, February 2013. [Online]. Available: http://tools.ietf.org/html/draft-dukkipati-tcpm-tcp-loss-probe

[50] R. project, "RTO-restart implementation for Linux 3.7." [Online]. Available: http://riteproject.eu/projects/wp1-end-systems-and-applications/rto-restart/

[51] F. Baker and G. Fairhurst, "IETF recommendations regarding active queue management," Internet Draft draft-ietf-aqm-recommendation, work in progress, Oct. 2013.

[52] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)," RFC 6817 (Experimental), Internet Engineering Task Force, Dec. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6817.txt

[53] L. Eggert and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers," RFC 5405 (Best Current Practice), Internet Engineering Task Force, Nov. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5405.txt

[54] G. Fairhurst, A. Sathiaseelan, and R. Secchi, "Updating TCP to support rate-limited traffic," Internet Draft draft-ietf-tcpm-newcwv-02, work in progress, Jul. 2013.

[55] ——, "Updating TCP to support rate-limited traffic," Internet Draft draft-ietf-tcpm-newcwv-02, work in progress, Jul. 2013.

[56] A. Sathiseelan, R. Secchi, G. Fairhurst, and I. Biswas, "Enhancing TCP to support rate-limited traffic," in *Capacity Sharing Workshop of ACM CoNEXT 2012*, Nice (France), Dec 2012.

[57] T. Kupka, P. Halvorsen, and C. Griwodz, "Performance of ON-OFF traffic stemming from live adaptive segmented HTTP video streaming," in *Proceedings of IEEE LCN*, Clearwater (US), 2012, pp. 401–409.

[58] I. Biswas, "Internet congestion control for variable rate TCP traffic," Ph.D. dissertation, University of Aberdeen, Aberdeen (UK), 2011.

[59] M. Handley, J. Padhye, and S. Floyd, "TCP Congestion Window Validation," RFC 2861 (Experimental), Internet Engineering Task Force, Jun. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2861.txt

[60] I. Biswas, A. Sathiaseelan, R. Secchi, and G. Fairhurst, "Analysing TCP for bursty traffic," *International Journal of Communications, Network and System Science*, vol. 7, no. 3, Jul 2010.

[61] "Network emulation (netem)," Linux Foundation, Nov 2009. [Online]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

[62] M. Yousaf, M. Welzl, and B. Yener, "Accurate shared bottleneck detection based on SVD and outlier detection," University of Innsbruck, Institute of Computer Science, Tech. Rep. DPS NSG Technical Report 1, August 2008.

[63] S. Hassayoun, J. Iyengar, and D. Ros, "Dynamic window coupling for multipath congestion control," in *Proceedings of IEEE ICNP*, oct. 2011, pp. 341–352.

[64] H. Balakrishnan, H. S. Rahul, and S. Seshan, "An integrated congestion management architecture for internet hosts," in *Proceedings of ACM SIGCOMM*. New York, NY, USA: ACM, 1999, pp. 175–187. [Online]. Available: http://doi.acm.org/10.1145/316188.316220

[65] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," RFC 5348 (Proposed Standard), Internet Engineering Task Force, sep 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5348.txt

[66] S.-H. Choi, "Congestion control for real-time interactive multimedia streams," Ph.D. dissertation, University College London, London, Sep. 2010. [Online]. Available: http://thesis.hackerslab.eu/

[67] M. Welzl, S. Islam, and S. Gjessing, "Coupled congestion control for RTP media," Internet Draft draft-welzl-rmcat-coupled-cc, work in progress, Oct. 2013. [Online]. Available: http://tools.ietf.org/html/draft-welzl-rmcat-coupled-cc

[68] A. Morton and B. Claise, "Packet Delay Variation Applicability Statement," RFC 5481 (Informational), Internet Engineering Task Force, Mar. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5481.txt

[69] ITU-T, "Recommendation Y.1540: Internet protocol data communication service – IP packet transfer and availability performance parameters," Mar. 2011. [Online]. Available: http://www.itu.int/rec/T-REC-Y.1540

[70] ——, "Recommendation Y.1541: Network performance objectives for IP-based services," Dec. 2011. [Online]. Available: http://www.itu.int/rec/T-REC-Y.1540

[71] R. Stewart, "Stream Control Transmission Protocol," RFC 4960 (Proposed Standard), Internet Engineering Task Force, Sep. 2007, updated by RFCs 6096, 6335. [Online]. Available: http://www.ietf.org/rfc/rfc4960.txt

[72] D. Wing and P. Natarajan, "Happy eyeballs: Trending towards success with SCTP," Internet Draft draft-wing-tsvwg-happy-eyeballs-sctp, work in progress, Oct. 2010. [Online]. Available: http://tools.ietf.org/html/draft-wing-tsvwg-happy-eyeballs-sctp

[73] M. Scharf and A. Ford, "Multipath TCP (MPTCP) Application Interface Considerations," RFC 6897 (Informational), Internet Engineering Task Force, Mar. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6897.txt

[74] T. Moncaster, M. Welzl, and D. Ros, "Problem statement: Why the IETF needs defined transport services," Internet Draft draft-moncaster-tsvwg-transport-services, work in progress, Oct. 2013. [Online]. Available: http://tools.ietf.org/html/draft-moncaster-tsvwg-transport-services

[75] C. Raiciu, M. Handley, and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols," RFC 6356 (Experimental), Internet Engineering Task Force, Oct. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6356.txt

[76] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J.-Y. Le Boudec, "MPTCP is not Pareto-optimal: performance issues and a possible solution," in *Proceedings of ACM CoNEXT*, Nice, Dec. 2012, pp. 1–12.

[77] R. Khalili, N. Gast, M. Popovic, and J.-Y. Le Boudec, "Opportunistic Linked-Increases Congestion Control Algorithm for MPTCP," Internet Draft draft-khalili-mptcp-congestion-control, work in progress, Feb. 2013.

[78] J. Iyengar, P. Amer, and R. Stewart, "Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths," *IEEE/ACM Transactions on Networking*, vol. 14, no. 5, pp. 951–964, Oct. 2006.

[79] Y.-C. Chen, Y.-S. Lim, R. Gibbens, E. Nahum, R. Khalili, and D. Towsley, "A measurement-based study of multipath TCP performance over wireless networks," in *Proceedings of the ACM Internet Measurement Conference (IMC'13)*, Barcelona, Oct. 2013.

[80] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 6824 (Experimental), Internet Engineering Task Force, Jan. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6824.txt

[81] T. Zinner, K. Tutschku, A. Nakao, and P. Tran-Gia, "Performance evaluation of packet re-ordering on concurrent multipath transmissions for transport virtualization," in *20th ITC Specialist Seminar on Network Virtualization – Concept and Performance Aspects*, Hoi An, Viet Nam, May 2009.

[82] C. Casetti, G. Galante, and R. Greco, "Load balancing over multipaths using bandwidth-aware source scheduling," in *7th International Symposium on Wireless Personal Multimedia Communications (WPMC 2004)*, Padova, Sep. 2004.

[83] M. Fiore and C. Casetti, "An adaptive transport protocol for balanced multihoming of real-time traffic," in *Proceedings of IEEE GLOBECOM*, Saint Louis (MO), USA, Nov. 2005, pp. 1091–1096.

[84] F. Perotto, C. Casetti, and G. Galante, "SCTP-based transport protocols for concurrent multipath transfer," in *Proceedings of IEEE WCNC*, Hong Kong, Mar. 2007, pp. 2971–2976.

[85] A. Gurtov and T. Polishchuk, "Secure multipath transport for legacy internet applications," in *Proceedings of BROADNETS'09*, Madrid, Sep. 2009.

[86] K. Evensen, D. Kaspar, A. Hansen, C. Griwodz, and P. Halvorsen, "Using multiple links to increase the performance of bandwidth-intensive UDP-based applications," in *Proceedings of IEEE ISCC*, Corfu, Greece, Jun. 2011, pp. 1117–1122.

[87] V. Sharma, S. Kalyanaraman, K. Kar, K. Ramakrishnan, and V. Subramanian, "MPLOT: A Transport Protocol Exploiting Multipath Diversity using Erasure Codes," in *Proceedings of IEEE INFOCOM*, Phoenix (AZ), USA, Apr. 2008.

[88] M. Kurant, "Exploiting the Path Propagation Time Differences in Multipath Transmission with FEC," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 5, pp. 1021–1031, May 2011.

[89] A. Vulimiri, O. Michel, P. Godfrey, and S. Shenker, "More is less: Reducing latency via redundancy," in *Eleventh ACM Workshop on Hot Topics in Networks (HotNets-XI)*, Redmond (WA), USA, 2012.

[90] B. Briscoe, A. Brunström, D. Ros, D. Hayes, A. Petlund, I.-J. Tsang, S. Gjessing, and G. Fairhurst, "A survey of latency reducing techniques and their merits," in *Internet Society Workshop on Reducing Internet Latency*, London, Sep. 2013. [Online]. Available: http://www.internetsociety.org/latency2013