ICT-317756

TRILOGY2

Trilogy2: Building the Liquid Net

Specific Targeted Research Project

FP7 ICT Objective 1.1 – The Network of the Future

D1.1 Software Platforms

Due date of deliverable: 31 December 2013

Actual submission date: 3 January 2014

Start date of project	1 January 2012
Duration	36 months
Lead contractor for this deliverable	Universitatea Politehnica Bucuresti (UPB)
Version	v1.0, January 3, 2014
Confidentiality status	Public

Abstract

This document captures the research carried in the first year of the Trilogy2 project related to the development of software platforms. It presents advances in the design, implementation and evaluation of tools which allow the creation of bandwidth, CPU and storage liquidity. Finally, the document concludes by describing the setup of a testbed that Trilogy2 partners are setting up in order to support liquidity experiments.

Target Audience

The target audience for this document is the networking research and development community, particularly those with an interest in Future Internet technologies and architectures. The material should be accessible to any reader with a background in network architectures, including mobile, wireless, service operator and data centre networks.

Impressum

Full project title	TRILOGY2: Building the Liquid Net
Title of the workpackage	D1.1 Software Platforms
Editor	Alexandru Agache, UPB
Project Co-ordinator	Marcelo Bagnulo Braun, UC3M
Copyright notice	© 2014 Participants in project TRILOGY2

Executive Summary

This document describes various software platforms developed for the Trilogy2 project which allow pooling of resources such as bandwidth, CPU and storage. Where possible, it also provides experimental results showing the extent to which liquidity can be created and deals with important topics such as performance and security.

In terms of bandwidth liquidity, we present two mechanisms, Multipath TCP and Polyversal TCP. The section on bandwidth liquidity starts by tackling three important aspects of Multipath TCP. The first is an evaluation of performance based on executing controlled experiments with MPTCP under the influence of factors such as capacity, queuing and propagation delay, loss and congestion control. The goal of these experiments is to verify if MPTCP can achieve its goals of improving throughput and balancing congestion. The results show this is possible and also provide further insight on the influence that various parameters and network phenomena have on MPTCP performance. The second deals with MPTCP security, and proposes a set of improvements. A residual threat analysis reveals a number of possible attacks, describes the prerequisite steps and information required in order to enact them, and establishes their relative threat level. Multiple security enhancements are proposed to address each shortcoming, and an outline describes the steps that will be taken as future work. The third aspect related to MPTCP is the exploration of novel use cases. We propose to use it to explicitly include middleboxes in MPTCP connections. This mechanism accommodates several widely encountered use cases, including load balancing, DDoS filtering and anycast services. We also show how MPTCP can be used in mobile devices to improve cellular connectivity.

Polyversal TCP (PVTCP), the next topic of this section, provides a mean to enable liquidity across different transports. It is based on the argument that sockets are an ossified and monolithic construct. They currently fail to take into account the actual context where communication takes place, and this has a less than ideal effect on entities running in tightly coupled contexts, such as processes which find themselves running on the same machine. A polyversal network stack seeks to select the best underlying communication mechanism, while shielding the application from unnecessary details. This allows PVTCP to pool the bandwidth of different available communication avenues not limited to a single transport.

We rely on virtualization as a key concept to achieve processing liquidity. We work on two flavors of CPU liquidity, one in mobiles and another one in network functions. The first mechanism for creating CPU liquidity presented in this document deals with virtual machine migration for mobile devices. They have been rapidly adopted by users but continue to lack a unified user experience. The goal is to transfer running apps and data between mobile devices running Android without changing the app state and preserving its network connectivity. The virtualization support can be used as a layer that separates data and applications from the environment of the mobile device. The other topic concerning CPU liquidity and efficient scaling is an example of turning specialized hardware middleboxes into one or multiple virtualized sofware devices that can run on inexpensive commodity hardware. We have used this paradigm for two key components of an ISP

network: the Broadband Remote Access Server (BRAS) and the Customer Provider Edge (CPE). A proof-of-concept implementation based on a lightweight Xen virtual machine for the BRAS case is presented, together with performance evaluation results.

The first mechanism that enables storage liquidity presented in this document is Irminsule, a Database-as-a-Service (DBaaS) solution focused on distributed storage, which presents a file-system layer solution for distributing data. It has the ability to checkpoint everything in independent branches and allows the rollback of entire distributed clusters. Storage liquidity is also enabled by Trevi, which uses fountain coding to tackle the distributed storage problem. Trevi represents a radical departure from what is currently the state of the art, by being free of TCP and incorporating fountain codes to multicast striped blobs across storage nodes. It does not require changes to the network stack or any network protocol. It is currently under implementation, and uses UDP or even raw Ethernet frames for network communication.

The final section of this document summarizes the implementation work done in the project and presents the testbed created with the concerted effort of multiple Trilogy2 partners. A description of the current hardware infrastructure is provided, together with details related to authorization and guidelines regarding the use of servers.

List of Authors

Authors	Alexandru Agache, Pedro Aranda, Mohamed Ahmed, Marcelo Bagnulo, Giacomo Bernini, Gino Carrozzo, Julian Chesterfield, Jon Crowcroft, Rareș Dumitrescu, Felipe Huici, Valentin Ilie, Anil Madhapeddy, Joao Martins, George Milescu, Cătălin Moraru, Dragoș Niculescu, Cătălin Nicuțar, Christoph Paasch, Costin Raiciu, John Thomson
Participants	Intel, NEC, NXW, OnApp, TID, UC3M, UCAM, UCL-BE, UPB
Work-package	WP1
Security	Public (PU)
Nature	R
Version	1.0
Total number of pages	128

Contents

List of Figures

List of Tables

Acronyms

BGP Border Gateway Protocol.

BRAS Broadband Remote Access Server.

CMS Cloud Management System.

CPE Customer Provider Edge.

DBaaS Database-as-a-Service.

DPI Deep Packet Inspection.

FTTH Fiber to the Home.

IaaS Infrastructure as a Service.

OVS Open virtual Switch.

PE Provider Edge.

PVTCP Polyversal TCP.

SDN Software Defined Networking.

VM Virtual Machine.

VPN Virtual Private Network.

VRF Virtual Routing and Forwarding.

1 Introduction

1.1 Motivation

The aim of the Trilogy2 project is to develop a new Internet architecture based on the concept of the liquid network. A liquid system should ideally allow resources including bandwidth, storage and processing to be used by any application, whether they are contributed by network operators, data centre operators or end systems. Resources form a shared pool and applications can scale up and down in multiple dimensions (storage, processing and bandwidth usage) as needed, in a continuous effort to enhance the users' experience as measured in terms of key metrics such as delay and battery life. The main objective of Trilogy2 is to unlock the value inherent in joining up the pools of liquidity in the Internet.

We seek to expand on current state-of-the-art techniques and systems for creating and controlling the liquidity of resources. In this document we describe building blocks of the Trilogy2 architecture that enable operators to create liquidity in their own networks. There are three principal resource areas where we seek to achieve and control liquidity: bandwidth, processing and storage. While some of the mechanisms described still represent work in progress, there are also instances where an implementation is already available, together with experimental results.

The Multipath TCP (MPTCP) protocol is a prime tool for creating bandwidth liquidity. While MPTCP was developed in the previous Trilogy project, there were still some open questions related to issues such as performance and security, and we want to offer satisfying answers for both. In addition, we also propose another tool for bandwidth liquidity, namely Polyversal TCP (PVTCP). As resources such as bandwidth become liquid we must also deal with situations that transcend in scope a particular transport such as TCP or MPTCP. While applications continue to use the same network abstraction we become interested in exploiting favorable circumstances regarding higher potential performance in the underlying implementation. This can be achieved with PVTCP.

The Trilogy2 project leverages virtualization as the primary mean to achieve CPU liquidity. We do so in two contexts i.e. using virtual machines in the mobile context and virtualization of network functions. Virtualization for mobile devices is an important setting to consider. Current mobile platforms offer CPUs capable of hardware virtualization that enable efficient use of the processing power. By transferring virtualized resources between mobile devices, running applications and data can be seamlessly moved across phones, tablets and PCs. Network Function Virtualization is an ongoing research effort to evolve from specialized hardware to provide network functions to software based solutions running in generic processing platforms. By moving to generic processing, it is possible to pool all the processing available in the network to perform different network functions. One especially attractive network function to virtualize is the BRAS due to its strategic position in the access network i.e. it is the IP element that is the closest to the customer. In the Trilogy 2 project we have worked in producing a virtualized BRAS (a.k.a. swBRAS) as part of our CPU liquidity

tool suite. In addition, we have also worked in providing a virtual CPE, another key component of the ISP network.

With an increasing amount of computing being performed on mobile devices that have less storage directly available and more consumers and enterprises using multiple devices to access data, there has been a rapid growth of cloud storage that allows access to the data across the Internet. The requirement for low latency services across the Internet has increased the need to place data efficiently and 'close' to the end-user. With greater user concerns of data privacy, who hosts the content and what access they have to the data is another metric that is becoming more prevalent. Controlling and managing storage as a resource with different constraints is therefore a real problem that entails many technical challenges. Trilogy2 will look at how storage can be managed and tied to computing resources.

1.2 A brief guide to the T2 architecture and the first set of deliverables

This deliverable describes the basic tools for the creation of liquidity, organized on the three types of resources: bandwidth liquidity, processing liquidity and storage liquidity.

Deliverable D1.2 describes the possible interactions between the different tools, as well with other elements of the network. They can be used independently or may serve as the basis on which different cross-resource liquidity scenarios.

Deliverable D2.1 describes a first iteration of the T2 architecture that encompasses all the described liquidity tools and their interactions.

Deliverable D2.2 describes the Trilogy 2 information model (represented in figure 1.1) that integrates the basic elements for liquidity. Each such mechanism falls into one of the three categories: bandwidth, processing (or CPU) and storage.

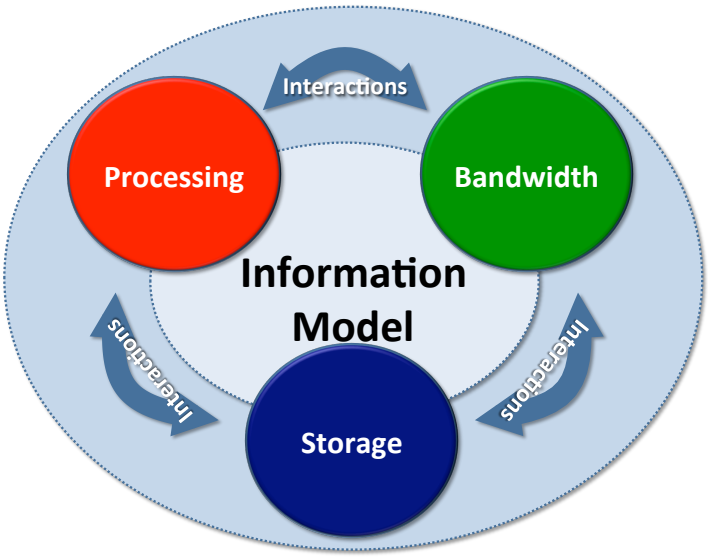


Figure 1.1: Trilogy 2 Information Model

1.3 Deliverable outline

This document is organized in the following manner.

In Section 2 we present the software platforms which allow pooling of bandwidth, MPTCP and PVTCP.

Section 3 provides an overview of two solutions that can be used for CPU and process pooling, virtual machine migration in mobile environments and the development of a swBRAS.

In Section 4 we describe Irminsule and Trevi, two mechanisms that deal with the pooling of storage resources in a distributed setting.

Finally, section 5 summarizes the work done towards the implementation of the prototypes and also offers more details about the setup of the experimental testbed that is used and supported by a number of partners.

2 Bandwidth Liquidity

This section describes the tools designed in Trilogy2 to allow bandwidth liquidity. We present two mechanisms that enable this liquidity, namely Multipath TCP and Polyversal TCP. Multipath TCP allows to pool the resources of different paths among the network in a single transport connection and Polyversal TCP is a novel proposal for generalized bandwidth pooling across multiple transports.

2.1 MPTCP

Multipath TCP (MPTCP) is an extension to TCP, proposed within the previous European FP7 Trilogy Project. MPTCP enables the pooling of multiple paths which results in bandwidth liquidity. It does so by allowing the concurrent use of multiple connections for the transmission of a single logical data-stream, without requiring any modifications among the applications and still being usable on today's Internet with all its middleboxes [48, 49]. MPTCP achieves this by creating multiple TCP subflows and multiplexing the data-segments among them[115]. This allows to pool the resources of every interface, effectively increasing the performance of the application. Further, MPTCP allows mobile nodes to vertically handover traffic from one interface to another, allowing smartphones to seamlessly move data connections from a WiFi access-point to the 3G connection [103]. The initial design and implementation of MPTCP was done in the previous Trilogy project and while the previous work has been very successful there is still work to be done in order to get MPTCP widely adopted. In particular, one of the challenges is to get MPTCP to become a Standard track document in the IETF. As of today, MPTCP is defined in a so-called experimental RFC. In order to get to a Standard track RFC, works needs to be done in order to prove that the MPTCP specification is mature and that there is a good understanding of the behaviour and possibilities of the protocol. The work done in the Trilogy2 project contributes to this effort. In particular, over this first year of the project, we have worked on three MPTCP related items. First we have performed an experimental evaluation of the MPTCP behaviour, which is necessary to grasp a full understanding of MPTCP. Second, we have worked in improving the security of MPTCP which is a mandatory step to move to the Standard track. Finally, we have investigated new possible uses of MPTCP in scenarios that are radically different than the ones envisioned in the original design.

2.1.1 Evaluating MPTCP's resource-pooling

An MPTCP implementation is a rather complex system. Many heuristics and algorithms influence its performance [115]. The congestion-control [152, 76] influences the sending-rate of the individual subflows, the scheduler decides how to multiplex data among the subflows, flow-control provides another limitation to the sending-rate, etc. Many external factors further influence the performance of MPTCP. Especially the network's characteristics in tems of capacity, propagation delay, etc. It is very difficult to have a clear understanding of how MPTCP behaves in these diverse environments. We apply the *Experimental Design* approach to evaluate the Linux Kernel implementation of MPTCP in a wide range of heterogeneous environments. We show that it is very beneficial to use such an approach, as it allowed us to reveal previously unknown perfor-

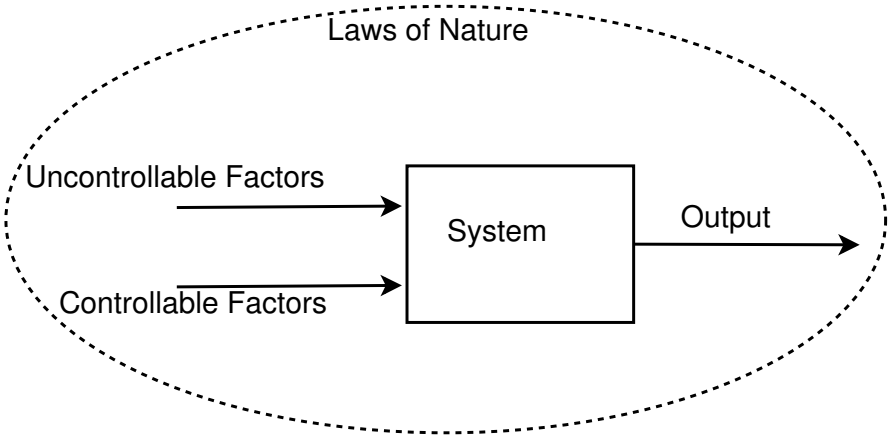


Figure 2.1: The system's output is influenced by a number of factors and obeys to the laws of nature [9]

mance issues within MPTCP. This section is based on work presented at the ACM CoNEXT conference in December 2013.

2.1.1.1 Experimental Design

Experimental Design refers to the process of executing controlled experiments in order to collect information about a specific process or system [44]. The system under experimentation is influenced by controllable or uncontrollable factors (see Figure 2.1). The system responds to these factors according to the laws of nature. The experimenter can observe these responses by collecting one or multiple outputs of the system, which provide the information necessary to understand the system.

Running experiments in computer science (and networking research) is peculiar in the sense that the output is often more deterministic (in absence of external factors, like temperature, time, etc.) [119]. Further, depending on the system, experiments may be cheap in terms of time required, thus allowing a large number of experiments. In this section, we give an overview of the different steps of experimental design and its particularities with respect to computer experiments.

2.1.1.1.1 Objective The first step is to define the objective pursued by the experiments. Kleijnen et al. defines in [77] three different types of questions that may be answered through experimentation:

- *"Develop a Basic Understanding"* - This allows to have an overview of the system's behavior, uncover problems within the system or confirm expectations.
- *"Finding Robust Decisions or Policies"* - The goal is to find the correct configuration of the system to produce a desired output, while taking into consideration the influence of uncontrollable factors.
- *"Comparing Decisions or Policies"* - This process enables us to predict the behavior of the system with respect to a specific set of factors.

The objective defines the system that is under test and the outcome we want to measure. Once the objective is defined, the influencing factors and the way of measuring the output of the system must be also defined.

2.1.1.1.2 Factors In experimental design, the system under test is often very complex. Many factors can influence the output of a system (see Figure 2.1). They may be of different kinds, controllable and uncontrollable. Among the controllable factors, those which influence the output of the system have to be selected. The uncontrollable factors may also influence the output of the system and are the reason for the variance of the system's output. To reduce the impact of the uncontrollable factors, an experiment should be repeated multiple times. This allows to extract the central tendency of the response by calculating the mean or median.

Optimal real-world experiments would require that the experiments are run simultaneously, to reduce the effect of the uncontrollable factors. However, in computer experiments this constraint can often be neglected as the output of the system is deterministic, allowing a sequential execution of the experiments [77].

2.1.1.1.3 Design of the experiment The design of the experiment influences the input parameters that are selected to conduct the experiments. In [18, 90], the desirable properties that such parameter sets should have are discussed.

If the behavior of the system is meant to be modeled by a first-order polynomial model, fractional designs are a good fit [18]. These designs distribute the parameter set along the edges. Further, orthogonality is a desirable criterion of experiment designs, as it ensures that the sets are uncorrelated and thus allows to decide if a factor should be part of the model or not [77].

However, sometimes it is not possible to assume a first-order polynomial model of the response. It may be that there is no prior knowledge of the system's response surface, or that the system has a rather stochastic nature. In this case, space-filling designs are good choices [88]. Space-filling designs don't only sample at the edges, but distribute the parameter sets equally among the whole factor space. A space-filling design can be generated with different algorithms. Santiago et al. propose in [122] the WSP algorithm which distributes the sets equally among the space. It is based on a uniform random sampling of the input parameters and eliminates excess points according to a minimum-distance criterion. The WSP algorithm has particularly good space-filling properties, even in high-dimensional spaces [122].

2.1.1.2 Experimenting with MPTCP

In this section we describe how the experimental design approach can be applied to the experimentation with a transport protocol like MPTCP. For this purpose, we need to define our objective, determine our design factors and experiment design.

2.1.1.2.1 Objective We are interested in a performance analysis of MPTCP to verify whether it fulfills its two main design goals [114]:

- *Improve throughput: MPTCP should perform at least as well as regular TCP along the best path.*
- *Balance congestion: MPTCP should move traffic away from congested paths.*

We evaluate the performance of MPTCP for a wide range of parameters and pinpoint the scenarios where these goals are not met. We can also use this framework to validate the performance of MPTCP as modifications to certain algorithms within the protocol are being done.

We execute our approach by using Mininet [61] which allows us to easily create a virtual network and run experiments between Mininet hosts using the v0.86 Linux Kernel implementation of MPTCP[1]. The benefit of using Mininet is that the results are reproducible and do not require a large number of physical machines. Compared to simulations, Mininet allows us to use the real MPTCP implementation and not a model of the protocol.

2.1.1.2.2 Factors The performance of a transport protocol like MPTCP is influenced by various factors, such as bandwidth limitations, propagation delay, queuing delay, loss, etc. [7]. Further, memory constraints on either of both hosts will limit the TCP window size, additionally influencing the performance [15, 127]. Among these factors, one must distinguish between the quantitative ones (e.g., loss-rate between two hosts) and the qualitative ones (e.g., congestion-control being used). For each of the selected factors, the domain must be selected accordingly. We consider the following factors in our study:

Capacity Our evaluation of MPTCP targets environments of regular users, whose access speed may range from mobile networks to Fiber to the Home (FTTH). We fix the range of our capacity from 0.1 to 100 Mbps.

Propagation delay The propagation delay is the round-trip-time between the sender and the receiver over an uncongested path. Measurement studies have shown that the delay may be up to 400 ms [157] . We set the delay to a domain between 0 ms and 400 ms.

Queuing delay The buffers at the bottleneck influence the queuing delay [53]. A perfect Active Queue Management (AQM) algorithm at the bottleneck router would not add any additional delay, whether a badly sized buffer may add a huge amount of queuing delay [53, 6]. We only consider tail-drop queues whose size is set to add a queuing delay between 0 ms up to 2000 ms. We leave the evaluation of different queuing policies like RED[47] or Codel[93] for future work.

Loss [92] shows that the loss probability over the Internet is very low (between 0 and 0.1%). In wireless or mobile networks, the loss probability may be considerably higher. We consider environments where the loss ranges from 0% to 2.5%.

Congestion Control We consider the two MPTCP congestion-control schemes: the Coupled congestion control [114, 152] which is the default one and the recently proposed OLIA [76].

These factors and their considered ranges allow to cover the principal environments that MPTCP might face when being used over the Internet. Additional factors and/or broader ranges are left for future work.

[1] Our scripts and the Mininet virtual images are available at http://multipath-tcp.org/conext2013

2.1.1.2.3 Design of the experiment We cannot be sure of the nature of the response surface of Multipath TCP. Hence, we choose a space-filling design to cover a wide range of scenarios and correlations among the factors. It allows us to avoid making any assumptions on the behavior of MPTCP (cfr. Section 2.1.1.1.3). The drawback is that we need to run a large number of experiments in order to fully cover the factor space, but thanks to Mininet we are able to quickly perform these experiments. We use the WSP algorithm to generate the parameter sets in the space-filling design.

2.1.1.3 Evaluation

This section shows the experiments we conducted and how they helped us to identify previously unknown performance issues with MPTCP.

2.1.1.3.1 Aggregation Benefit We study whether MPTCP satisfies its first design goal (improve throughput), by quantifying the aggregation benefit (as defined by Kaspar in [74]). The aggregation benefit is expressed as a function between -1 and 1. If MPTCP performs as well as the path with the highest goodput, the aggregation benefit will be 0. If MPTCP perfectly aggregates the capacities of all paths, the aggregation benefit will be 1. -1 means that MPTCP achieves zero goodput.

Let S be a multipath aggregation scenario, with n paths. C_i is the capacity of the path i and C_{max} the highest capacity among all paths. If we measure a goodput of g with MPTCP, the aggregation benefit, $Ben(S)$, is given by [74]:

$$Ben(S) = \begin{cases} \dfrac{g - C_{max}}{\sum_{i=1}^{n} C_i - C_{max}}, & \text{if } g \geq C_{max} \\ \dfrac{g - C_{max}}{C_{max}}, & \text{if } g < C_{max} \end{cases}$$

Our setup evaluates MPTCP in a scenario (Figure 2.2) where the hosts establish two subflows between each other. We consider this as the common scenario (e.g., a client having two access networks like WiFi/3G). In order to measure the aggregation benefit, the Mininet-hosts create an iperf-session using the v0.86-release of MPTCP, which creates one subflow per bottleneck-link. The iperf-session runs for 60 seconds to allow the flows to reach equilibrium.

We study two types of environments: low Bandwidth-Delay-Products (BDP) and high-BDP. Low-BDP environments have relatively small propagation and queuing delays. In a high-BDP environment, the maximum values for the propagation and the queuing delays are very large. In a first run we only consider 3 factors per bottleneck, namely the capacity, propagation delay and queuing delay. For this first run we do not add the loss-factor as the MPTCP-specific congestion controls have a very specific behavior in lossy environments (as can be seen at the end of this section). The exact specifications of each environment can be found in Table 2.1.

As we consider 2 paths, each being influenced by 3 factors, we have a 6-dimensional parameter space. We generate the parameter sets by using the WSP space-filling design, resulting in about 200 individual experiments. We have limited ourselves to 200 experiments, as our Mininet environment is able to run these within 4 hours. This allows us to quickly obtain the results of the experiments. In order to cope with possible

	Low-BDP		High-BDP	
Factor	Min.	Max.	Min.	Max.
Capacity [Mbps]	0.1	100	0.1	100
Propagation delay [ms]	0	50	0	400
Queuing [ms]	0	100	0	2000

Table 2.1: Domains of the influencing factors for the measurement of aggregation benefit.

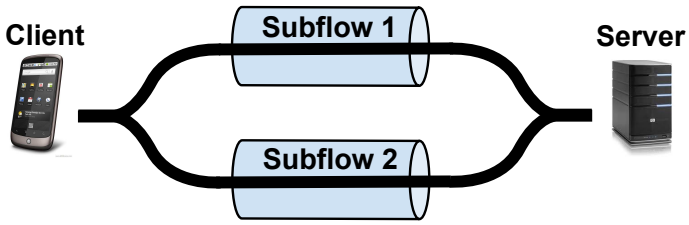


Figure 2.2: Our topology to evaluate aggregation benefit: MPTCP creates one subflow across each bottleneck. This allows to evaluate multipath-scenarios like a mobile phone connecting to WiFi/3G at the same time.

variations, we repeat each parameter set 5 times and use the median to extract the central tendency of the aggregation benefit.

2.1.1.3.1.1 Effect of receive-buffer sizes The performance of MPTCP is influenced by the receive-buffer sizes of the end hosts [115]. We evaluate the impact of a fixed receive buffer on the aggregation benefit in the low-BDP and the high-BDP environments. In Figure 2.3 we show the aggregation benefit's mean (with its standard deviation) and the median, 25% and 75% percentiles as well as the degree of dispersion. We see that the larger the receive buffer, the larger the aggregation benefit is. If the receive buffer is small, the MPTCP session is flow limited and thus cannot use the full capacity of the links, which reduces the aggregation benefit. The results are shown when coupled congestion control [152] is used. In this scenario, OLIA [76] performs similarly to coupled congestion control.

2.1.1.3.1.2 Effect of enabling auto-tuning The Linux TCP stack does not use a fixed receive buffer. Instead, it includes an auto-tuning algorithm [127] that adapts dynamically to the size of the receive buffer to achieve high goodput at the lowest possible memory cost. The current MPTCP implementation sets the

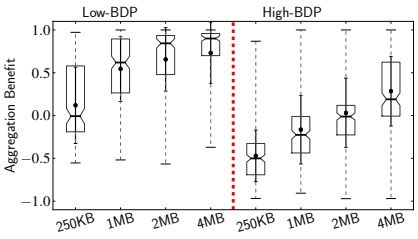


Figure 2.3: The aggregation benefit increases with the buffer size. If the buffer is small, MPTCP is limited by the receive window and does not fully utilizes the network's capacity.

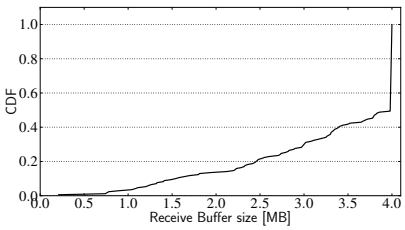


Figure 2.4: With auto-tuning, 50% of the experiments are able to consume less than 4MB of receive buffer in the low-BDP environment.

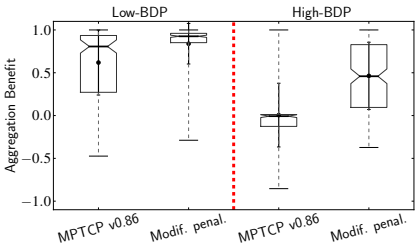


Figure 2.5: MPTCP v0.86 has a weak performance with auto-tuning. Our modification to the penalization algorithm significantly improves the aggregation benefit.

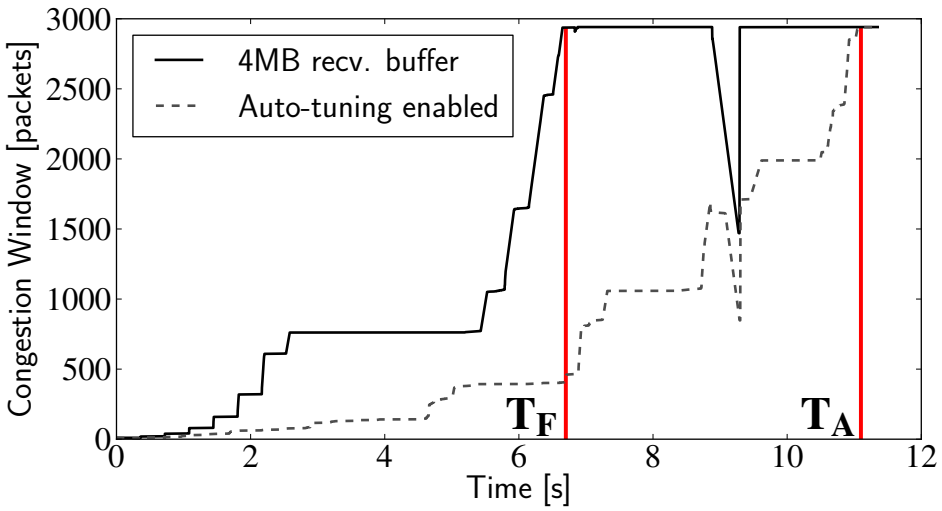


Figure 2.6: During the slow-start phase of an MPTCP subflow, the congestion window increases much slower when auto-tuning is enabled.

buffer to $2 * \sum_i^n bw_i * RTT_{max}$, to allow aggregating the throughput of all the subflows [15].

Enabling auto-tuning should reduce the memory requirements of the MPTCP connection. Figure 2.4 reports the maximum receive buffer used in the low-BDP environment for each set. Indeed, we observe that on 50% of the experiments the receive buffer remains below 4 MB, effectively reducing the memory used by the connection.

However, enabling auto-tuning can also lead to a huge performance degradation with MPTCP. Figure 2.5 depicts the aggregate benefit of MPTCP v0.86 when auto-tuning is enabled, capping the buffer to Linux's default of 4 MB. We observe that in high-BDP environments the aggregation benefit is significantly smaller than if the receive buffer is fixed at 4 MB (cfr. Figure 2.4). Effectively, 80% of the experiments have an aggregation benefit below 0. We observe this performance degradation in low-BDP environments too: 25% of the experiments have an aggregation benefit below 0.1.

The auto-tuning at the receiver will make the receive buffer start at a small value at the beginning of the connection, increasing it as the sender's subflows evolve during their slow-start phase. MPTCP evaluates the receive-buffer size every RTT_{max} in order to estimate the sending rate of the peer. $\sum_i^n bw_i * RTT_{max}$ represents the amount of data the peer sends during one RTT_{max}-interval. Multiplying this by 2 to achieve the recommended receive buffer of [15] should allow the sender to increase its sending rate during the slow-start phase. However, subflows whose RTT is smaller than RTT_{max} will more than double their sending rate during an RTT_{max}-interval, effectively making the sender limited by the receive window. As the subflows evolve through their slow-start phase, the announced window will continue increasing and eventually be large enough. Hence, these receive-window limitations are only transient and should not prevent users from achieving a high transmission rate.

MPTCP's reaction to transient receive-window limitations is overly aggressive because of the *"penalization"*

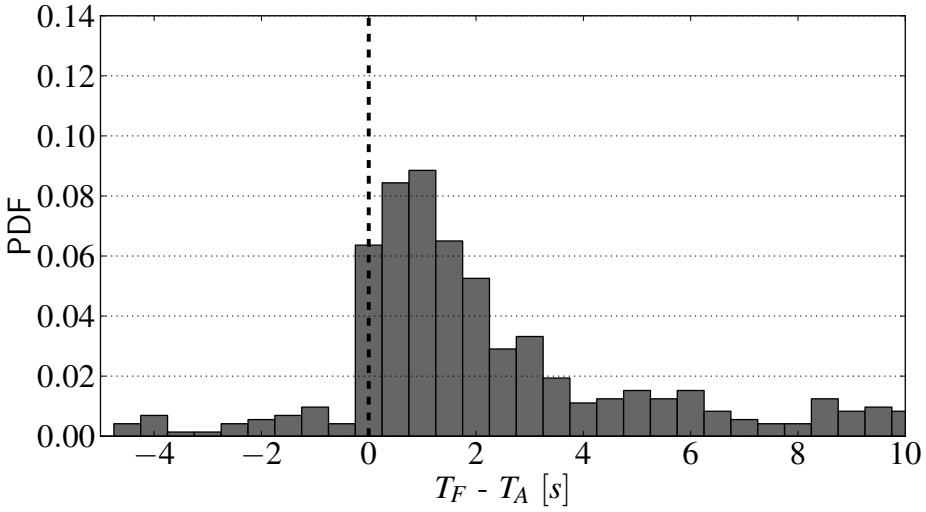


Figure 2.7: With auto-tuning enabled, MPTCP is slower to increase the congestion window during slow-start compared to a fixed receive buffer at 4 MB.

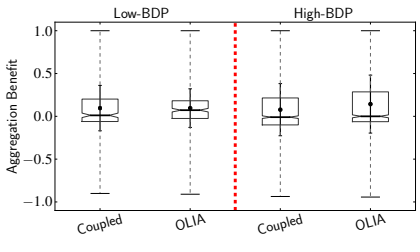


Figure 2.8: The MPTCP congestion controls move traffic away from congested (aka lossy) paths. Only the less-lossy path is used for MPTCP.

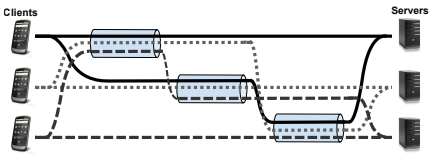


Figure 2.9: Three bottlenecks are used to evaluate MPTCP's load-balancing performance. Each MPTCP session has a one-hop and a two-hop subflow.

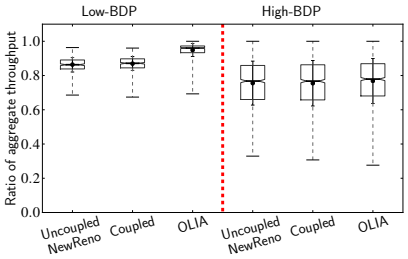


Figure 2.10: In the low-BDP environments, OLIA is able to efficiently move the traffic away from the congested paths.

mechanism proposed in [115]. This mechanism handles receive-window limitations due to round-trip-time differences among the subflows (e.g., in WiFi/3G environments). When the flow is limited by the receive window, it halves the congestion window of the subflow who causes this receive-window limitation and sets its slow-start threshold to the current congestion window. If an MPTCP-connection experiences the transient receive-window limitations while one of its subflows is in slow-start, the penalization algorithm will give this subflow a false view of the path capacity by adjusting its slow-start threshold to a smaller value.

We propose to modify the penalization algorithm. It should not adjust the slow-start threshold when a subflow is in its slow-start phase. Figure 2.5 shows how this small modification to the penalization algorithm improves the performance of MPTCP. In the low-BDP environments, more than 75% of the experiments achieve an aggregation benefit of 0.85 or higher. And even the experiments in the high-BDP environments have their median increased up to an aggregation benefit of around 0.5. Long data transfers with MPTCP are now less vulnerable to transient receive-window limitations and achieve a high aggregation benefit.

2.1.1.3.1.3 Effect of transient receive-window limitations Our modification to the penalization algorithm mitigates the effect of the transient receive-window limitations for long flows. However, the fundamental problem is still there. Transient receive-window limitations slow down the increase rate of the congestion window during slow-start. We picked a specific set of parameters where these transient receive-window limitations are particularly apparent. The first bottleneck has a capacity of 1 Mbps, while the second has a capacity of 50 Mbps. Both bottlenecks have a high propagation delay and queueing delay. A plot of the congestion-window's evolution of the fastest path during the slow-start phase is shown in Figure 2.6[2]. When auto-tuning is enabled, the congestion window increases much slower, reaching its maximum at time T_A, about four seconds later compared to a fixed receive buffer at 4 MB (T_F).

We measure the difference between T_A and T_F in our high-BDP environment among the 200 parameter sets generated with the space-filling design. Figure 2.7 illustrates these differences. Ideally, the difference between T_A and T_F should be zero. However, this is only true for 6% of the experiments. For a large portion of the experiments the congestion window reaches its maximum between 500ms and 2000ms faster if auto-tuning is disabled.

This could have a negative impact on flow-completion time of short flows, as the connection is receive-window limited during its slow-start phase. An ideal auto-tuning algorithm should not prevent the sender from increasing the sending rate - even during slow-start.

2.1.1.3.1.4 Effect of losses In this section, we study the effect of transmission losses on the performance of MPTCP. This could represent wireless environments where losses occur due to the fading. We model lossy links by adding a loss factor to each bottleneck, effectively increasing the parameter space to a total of 8 factors. These loss-factors range from 0 to 2.5% and are set individually on each bottleneck. The two MPTCP congestion controls (Coupled and OLIA [152, 76]) both try to move traffic away from congested paths. As these loss-based congestion controls interpret a loss as congestion, they move traffic away from lossy subflows. Figure 2.8 shows that Coupled and OLIA have mostly an aggregation benefit of 0. This confirms that Coupled and OLIA only push traffic on the less lossy of the two subflows, thus moving almost all traffic to the best path.

2.1.1.3.2 Congestion Balancing In this section, we analyze whether MPTCP satisfies its congestion-balancing design goal. For this purpose, we study the performance of MPTCP in the scenario of Figure 2.9. The network contains three bottlenecks, and the end-hosts create a total of three MPTCP sessions passing by this network. Each session creates two subflows, one crossing a single-bottleneck and the other passing through two bottlenecks. As discussed in [152], to balance the congestion and hence maximize the throughput for all MPTCP sessions, no traffic should be transmitted over the two-hop subflows. The bottlenecks are influenced by the capacity, propagation delay and queuing delay, effectively emulating the low-BDP and

[2]The Figure shows that even the flow whose receive buffer is fixed at 4 MB experiences a stall in the congestion-window's increase rate (between 2.5 and 5.5 seconds). This is due to a very specific issue of the Linux TCP window handling and has been fixed by http://patchwork.ozlabs.org/patch/273311/. Unfortunately the Linux MPTCP-stack does not yet include this recent fix.

high-BDP environments form Table 2.1. With three bottlenecks and three factors per bottleneck, we have effectively a 9-dimensional parameter space. We generate about 400 parameter sets with the WSP space-filling algorithm and start iperf-sessions for each MPTCP session.

We evaluate this scenario and show the relation between the aggregated goodput of all MPTCP sessions, compared to the theoretical upper bound in Figure 2.10. We compare the performance of OLIA and Coupled congestion control with the case that uncoupled NewReno congestion control[3] is used. We observe that OLIA is able to move traffic away from the two-hop subflows in low-BDP environments and hence efficiently uses the capacity available in the network. Coupled fails to provide any load balancing in the network and performs similarly to uncoupled NewReno. This observation confirms previous discussions about performance issues with coupled congestion control [76]. In high-BDP environments, even OLIA fails to provide a good congestion balancing. This is because the large BDP makes one of the three MPTCP sessions become receive-window limited if the three bottlenecks have different characteristics. This flow cannot benefit of OLIA's load-balancing algorithms and leads to suboptimal network utilization.

2.1.1.4 Sensitivity Analysis

The number of experiments executed within the parameter space influences the accuracy of the results. The more sets explored, the better the accuracy will be. However, CPU-time constraints limit the number of experiments that can be executed. The sensitivity analysis allows to confirm that the number of experiments conducted per parameter space is sufficient to have an accurate view of MPTCP's performance. This can be achieved by generating different space-filling designs, and comparing the 5th, 25th, 75th and 95th percentiles and median among each of these designs.

To evaluate the aggregation-benefit we used three influencing factors per bottleneck link, effectively creating a 6-dimensional parameter space. 200 sets were generated, using the WSP space-filling algorithm. We generate 5 different space-filling designs of comparable size for the sensitivity analysis. Each design explores different sets among the parameter space. Comparing the percentiles and the median of the aggregation benefit for each of the designs has shown that the standard deviation is very low. Relative to the range of the aggregation benefit $(-1, 1)$, it ranges between 0.1% and 2.62%. We can conclude that running 200 experiments is sufficient to have a good overview of the aggregation benefit of MPTCP in our 6-dimensional parameter space. In the load-balancing environment we observe a similarly low standard deviation ranging from 0.008% to 1.4%.

2.1.2 MPTCP security improvements

The goal of this line of work is to improve MPTCP security. In order to do that, we first perform an extensive threat analysis that will enable us to identify the required countermeasures. This section contains this first step, the residual threat analysis. RFC 6181 [10] provided a threat analysis for the general solution space of extending TCP to operate with multiple IP addresses per connection. Its main goal was to leverage previous

[3] Uncoupled NewReno [62] represents the case where regular TCP congestion control is used on the subflows. It increases the congestion windows of each subflow irregardless of the congestion state of the other subflows that are part of the MPTCP session.

experience acquired during the design of other multi-address protocols, notably SHIM6 [96], SCTP [132] and MIPv6 [73] during the design of MPTCP. Thus, RFC 6181 was produced before the actual MPTCP specification [49] was completed, and documented a set of recommendations that were considered during the production of such specification.

This analysis complements RFC 6181 with a vulnerability analysis of the specific mechanisms specified in RFC 6824. The motivation is to identify possible security issues with MPTCP as currently specified and propose security enhancements to address the identified security issues.

The goal of the security mechanisms defined in RFC 6824 were to make MPTCP no worse than currently available single-path TCP. We believe that this goal is still valid, so we will perform our analysis on the same grounds.

Types of attackers: for all attacks considered, we identify the type of attacker. We can classify the attackers based on their location as follows:

- Off-path attacker. This is an attacker that does not need to be located in any of the paths of the MPTCP session at any point in time during the lifetime of the MPTCP session. This means that the Off-path attacker cannot eavesdrop any of the packets of the MPTCP session.

- Partial time On-path attacker. This is an attacker that needs to be in at least one of the paths during part but not during the entire lifetime of the MPTCP session. The attacker can be in the forward and/or backward directions, for the initial subflow and/or other subflows. The specific needs of the attacker will be made explicit in the attack description.

- On-path attacker. This attacker needs to be on at least one of the paths during the whole duration of the MPTCP session. The attacker can be in the forward and/or backward directions, for the initial subflow and/or other subflows. The specific needs of the attacker will be made explicit in the attack description.

We can also classify the attackers based on their actions as follows:

- Eavesdropper. The attacker is able to capture some of the packets of the MPTCP session to perform the attack, but it is not capable of changing, discarding or delaying any packet of the MPTCP session. The attacker can be in the forward and/or backward directions, for the initial subflow and/or other subflows. The specific needs of the attacker will be made explicit in the attack description.

- Active attacker. The attacker is able to change, discard or delay some of the packets of the MPTCP session. The attacker can be in the forward and/or backward directions, for the initial subflow and/or other subflows. The specific needs of the attacker will be made explicit in the attack description.

We consider the following possible combinations of attackers:

- an On-path eavesdropper

- an On-path active attacker
- an Off-path active attacker
- a Partial-time On-path eavesdropper
- a Partial-time On-path active attacker

In the rest of this section we describe different attacks that are possible against the MPTCP protocol specified in RFC6824 and we propose possible security enhancements to address them.

2.1.2.1 ADD_ADDR attack

2.1.2.1.1 Summary of the attack:

Type of attack: MPTCP session hijack enabling Man-in-the-Middle.

Type of attacker: Off-path, active attacker.

Threat: Medium

2.1.2.1.2 Description In this attack, the attacker uses the ADD_ADDR option defined in RFC6824 to hijack an ongoing MPTCP session and enables himself to perform a Man-in-the-Middle attack on the MPTCP session.

Consider the following scenario. Host A with address IPA has one MPTCP session with Host B with address IPB. The MPTCP subflow between IPA and IPB is using port PA on host A and port PB on host B. The tokens for the MPTCP session are TA and TB for Host A and Host B respectively. Host C is the attacker. It owns address IPC. The attack is executed as follows:

(i) Host C sends a forged packet with source address IPA, destination address IPB, source port PA and destination port PB. The packet has the ACK flag set. The TCP sequence number for the segment is i and the ACK sequence number is j. We will assume all these are valid, we discuss what the attacker needs to figure these ones later on. The packet contains the ADD_ADDR option. The ADD_ADDR option announces IPC as an alternative address for the connection. It also contains an eight bit address identifier which does not bring any strong security benefit.

(ii) Host B receives the ADD_ADDR message and it replies by sending a TCP SYN packet. (Note: the MPTCP specification states that the host receiving the ADD_ADDR option may initiate a new subflow. If the host is configured so that it does not initiate a new subflow the attack will not succeed. For example, on the Linux implementation, the server does not create subflows. Only the client does so.) The source address for the packet is IPB, the destination address for the packet is IPC, the source port is PB' and the destination port is PA' (It is not required that PA=PA' nor that PB=PB'). The sequence number for this packet is the new initial sequence number for this subflow. The ACK sequence number

is not relevant as the ACK flag is not set. The packet carries an MP_JOIN option and it carries the token TA. It also carries a random nonce generated by Host B called RB.

(iii) Host C receives the SYN+MP_JOIN packet from Host B, and it alters it in the following way. It changes the source address to IPC and the destination address to IPA. It sends the modified packet to Host A, impersonating Host B.

(iv) Host A receives the SYN+MP_JOIN message and it replies with a SYN/ACK+MP_JOIN message. The packet has source address IPA and destination address IPC, as well as all the other needed parameters. In particular, Host A computes a valid HMAC and places it in the MP_JOIN option.

(v) Host C receives the SYN/ACK+MP_JOIN message and it changes the source address to IPC and the destination address to IPB. It sends the modified packet to IPB impersonating Host A.

(vi) Host B receives the SYN/ACK+MP_JOIN message. Host B verifies the HMAC of the MP_JOIN option and confirms its validity. It replies with an ACK+MP_JOIN packet. The packet has source address IPB and destination address IPC, as well as all the other needed parameters. The returned MP_JOIN option contains a valid HMAC computed by Host B.

(vii) Host C receives the ACK+MP_JOIN message from B and it alters it in the following way. It changes the source address to IPC and the destination address to IPA. It sends the modified packet to Host A impersonating Host B.

(viii) Host A receives the ACK+MP_JOIN message and creates the new subflow.

At this point the attacker has managed to place itself as a MitM for one subflow for the existing MPTCP session. It should be noted that there still exists the subflow between address IPA and IPB that does not flow through the attacker, so the attacker has not completely intercepted all the packets in the communication (yet). If the attacker wishes to completely intercept the MPTCP session it can do the following additional step.

(ix) Host C sends two TCP RST messages. One TCP RST packet is sent to Host B, with source address IPA and destination address IPB and source and destination ports PA and PB, respectively. The other TCP RST message is sent to Host A, with source address IPB and destination address IPA and source and destination ports PB and PA, respectively. Both RST messages must contain a valid sequence number. Note that figuring the sequence numbers to be used here for subflow A is the same difficulty as being able to send the initial ADD_ADDR option with valid Sequence number and ACK value. If there are more subflows, then the attacker needs to find the Sequence Number and ACK for each subflow.

At this point the attacker has managed to fully hijack the MPTCP session.

2.1.2.1.3 Information required by the attacker to perform the described attack In order to perform this attack the attacker needs to guess or know the following pieces of information: (The attacker need this information for one of the subflows belonging to the MPTCP session.)

- the four-tuple Client-side IP Address, Client-side Port, Server- side Address, Servcer-side Port that identifies the target TCP connection
- a valid sequence number for the subflow
- a valid ACK sequence number for the subflow
- a valid address identifier for IPC

TCP connections are uniquely identified by the four-tuple Source Address, Source Port, Destination Address, Destination Port. Thus, in order to attack a TCP connection, an attacker needs to know or be able to guess each of the values in that four-tuple. Assuming the two peers of the target TCP connection are known, the Source Address and the Destination Address can be assumed to be known.

We note that in order to be able to successfully perform this attack, the attacker needs to be able to send packets with a forged source address. This means that the attacker cannot be located in a network where techniques like ingress filtering [43] or source address validation [155] are deployed. However, ingress filtering is not as widely implemented as one would expect, and hence cannot be relied upon as a mitigation for this kind of attack.

Assuming the attacker knows the application protocol for which the TCP connection is being employed, the server-side port can also be assumed to be known. Finally, the client-side port will generally not be known, and will need to be guessed by the attacker. The chances of an attacker guessing the client-side port will depend on the ephemeral port range employed by the client, and whether the client implements port randomization.

Assuming TCP sequence number randomization is in place, an attacker would have to blindly guess a valid TCP sequence number. That is,

$RCV.NXT \leq SEG.SEQ < RCV.NXT + RCV.WND$ or $RCV.NXT \leq SEG.SEQ + SEG.LEN - 1 < RCV.NXT + RCV.WND$

As a result, the chances of an attacker to succeed will depend on the TCP receive window size at the target TCP peer.

We note that automatic TCP buffer tuning mechanisms have been become common for popular TCP implementations, and hence very large TCP window sizes of values up to 2 MB could end up being employed by such TCP implementations.

The Acknowledgement Number is considered valid as long as it does not acknowledge the receipt of data that has not yet been sent. That is, the following expression must be true:

$SEG.ACK \leq SND.NXT$

Finally, in order for the address identifier to be valid, the only requirement is that it needs to be different than the ones already being used by Host A in that MPTCP session, so a random identifier is likely to work.

Given that a large number of factors affect the chances of an attacker of successfully performing the aforementioned off-path attacks, we provide a general expressions for the expected number of packets the attacker needs to send to succeed in the attack.

Packets = $(2^{32}/(RCV\_WND)) * 2 * EPH\_PORT\_SIZE/2 * 1/MSS$

Where the new :

Packets: Maximum number of packets required to successfully perform an off- path (blind) attack.

RCV_WND: TCP receive window size (RCV.WND) at the target node.

EPH_PORT_SIZE: Number of ports comprising the ephemeral port range at the client system.

MSS: Maximum Segment Size, assuming the attacker will send full segments to maximize the chances to get a hit.

In order to have some idea of the magnitude of the amount of packets required, for an ephemeral port range of 4000 ports and using packets of 16KB, an attacker would need to send 699050 packets to perform the attack. While this is not trivial to achieve, it is not unfeasible neither, so, we conclude that the attack should be taken into consideration.

2.1.2.1.4 Possible security enhancements to prevent this attack

- To include the token of the connection in the ADD_ADDR option. This would make it harder for the attacker to launch the attack, since he needs to either eavesdrop the token (so this can no longer be a blind attack) or to guess it, but a random 32 bit number is not so easy to guess. However, this would imply that any eavesdropper that is able to see the token, would be able to launch this attack. This solution then increases the vulnerability window against eavesdroppers from the initial 3-way handshake for the MPTCP session to any exchange of the ADD_ADDR messages.

- To include the HMAC of the address contained in the ADD_ADDR option concatenated with the key of the receiver of the ADD-ADDR message. This makes it much more secure, since it requires the attacker to have both keys (either by eavesdropping it in the first exchange or by guessing it). Because this solution relies on the key used in the MPTCP session, the protection of this solution would increase if new key generation methods are defined for MPTCP (e.g. using SSL keys as has been proposed).

- To include the destination address of the ADD_ADDR message in the HMAC. This would certainly make the attack harder (the attacker would need to know the key). It wouldn't allow hosts behind NATs to be reached by an address in the ADD_ADDR option, even with static NAT bindings (like a web server at home). Probably it would make sense to combine it with option 2) (i.e. to have the HMAC of the address in the ADD_ADDR option and the destination address of the packet).

- To include the destination address of the SYN packet in the HMAC of the MP_JOIN message. This has the same problems than option 3) in the presence of NATs.

2.1.2.2 DoS attack on MP_JOIN

2.1.2.2.1 Summary of the attack:

Type of attack: MPTCP Denial-of-Service attack, preventing the hosts from creating new subflows.

Type of attacker: Off-path, active attacker

Threat: Low (as it is hard to guess the 32-bit token and still then the attacker only prevents the creation of new subflows)

2.1.2.2.2 Description: As currently specified, the initial SYN+MP_JOIN message of the 3-way handshake for additional subflows creates state in the host receiving the message. This, because the SYN+MP_JOIN contains the 32-bit token that allows the receiver to identify the MPTCP-session and the 32-bit random nonce, used in the HMAC calculation. As this information is not resent in the third ACK of the 3-way handshake, a host must create state upon reception of a SYN+MP_JOIN.
Assume that there exists an MPTCP-session between host A and host B, with token Ta and Tb. An attacker, sending a SYN+MP_JOIN to host B, with the valid token Tb, will trigger the creation of state on host B. The number of these half-open connections a host can store per MPTCP-session is limited by a certain number, and it is implementation-dependent. The attacker can simply exhaust this limit by sending multiple SYN+MP_JOINs with different 5-tuples. The (possibly forged) source address of the attack packets will typically correspond to an address that is not in use, or else the SYN/ACK sent by Host B would elicit a RST from the impersonated node, thus removing the corresponding state at Host B.
This effectively prevents the host A from sending any more SYN+ MP_JOINs to host B, as the number of acceptable half-open connections per MPTCP-session on host B has been exhausted.
The attacker needs to know the token Tb in order to perform the described attack. This can be achieved if it is a partial on-time eavesdropper, observing the 3-way handshake of the establishment of an additional subflow between host A and host B. If the attacker is never on-path, it has to guess the 32-bit token.

2.1.2.2.3 Possible security enhancements to prevent this attack The third packet of the 3-way handshake could be extended to contain also the 32-bit token and the random nonce that has been sent in the SYN+MP_JOIN. Further, host B will have to generate its own random nonce in a reproducible fashion (e.g., a Hash of the 5-tuple + initial sequence-number + local secret). This will allow host B to reply to a SYN+MP_JOIN without having to create state. Upon the reception of the third ACK, host B can then verify the correctness of the HMAC and create the state.

2.1.2.3 SYN flooding amplification

2.1.2.3.1 Summary of the attack:

Type of attack: The attacker can use the SYN+MP_JOIN messages to amplify the SYN flooding attack.

Type of attacker: Off-path, active attacker

Threat: Medium

2.1.2.3.2 Description: SYN flooding attacks use SYN messages to exhaust the server's resources and prevent new TCP connections. A common mitigation is the use of SYN cookies that allow the stateless processing of the initial SYN message.
With MPTCP, the initial SYN can be processed in a stateless fashion using the aforementioned SYN cookies. However, as we described in the previous section, as currently specified, the SYN+MP_JOIN messages are not processed in a stateless manner. This opens a new attack vector. The attacker can now open a MPTCP session by sending a regular SYN and creating the associated state but then send as many SYN+MP_JOIN messages as supported by the server with different source address source port combinations, consuming server's resources without having to create state in the attacker. This is an amplification attack, where the cost on the attacker side is only the cost of the state associated with the initial SYN while the cost on the server side is the state for the initial SYN plus all the state associated to all the following SYN+MP_JOIN.

2.1.2.3.3 Possible security enhancements to prevent this attack

(i) The solution described for the previous DoS attack on MP_JOIN would also prevent this attack.

(ii) Limiting the number of half open subflows to a low number (in the order of 3) would also limit the impact of this attack.

2.1.2.4 Eavesdropper in the initial handshake

2.1.2.4.1 Summary of the attack:

Type of attack: An eavesdropper present in the initial handshake where the keys are exchanged can hijack the MPTCP session at any time in the future.

Type of attacker: a Partial-time On-path eavesdropper

Threat: Low

2.1.2.4.2 Description: In this case, the attacker is present along the path when the initial 3-way handshake takes place, and therefore is able to learn the keys used in the MPTCP session. This allows the attacker to move away from the MPTCP session path and still be able to hijack the MPTCP session in the future. This vulnerability was readily identified at the moment of the design of the MPTCP security solution and the threat was considered acceptable.

2.1.2.4.3 Possible security enhancements to prevent this attack

There are many techniques that can be used to prevent this attack and each of them represents different tradeoffs. At this point, we limit ourselves to enumerate them and provide useful pointers.

(i) Use of hash-chains. The use of hash chains for MPTCP has been explored in [34]

(ii) Use of SSL keys for MPTCP security as described in [102]

(iii) Use of Cryptographically-Generated Addresses (CGAs) for MPTCP security. CGAs have been used in the past to secure multi addressed protocols like SHIM6 [96].

(iv) Use of TCPCrypt [17]

2.1.2.5 SYN/JOIN attack

2.1.2.5.1 Summary of the attack:

Type of attack: An attacker that can intercept the SYN/JOIN message can alter the source address being added.

Type of attacker: a Partial-time On-path eavesdropper

Threat: Low

2.1.2.5.2 Description:

The attacker is present along the path when the SYN/JOIN exchange takes place, and this allows the attacker to add any new address it wants to by simply substituting the source address of the SYN/JOIN packet for one it chooses. This vulnerability was readily identified at the moment of the design of the MPTCP security solution and the threat was considered acceptable.

2.1.2.5.3 Possible security enhancements to prevent this attack

It should be noted that this vulnerability is fundamental due to the NAT support requirement. In other words, MPTCP MUST work through NATS in order to be deployable in the current Internet. NAT behavior is unfortunately indistinguishable from this attack. It is impossible to secure the source address, since doing so would prevent MPTCP to work though NATS. This basically implies that the solution cannot rely on securing the address. A more promising approach would be then to look into securing the payload exchanged, limiting the impact that the attack would have in the communication (e.g. TCPCrypt or similar).

2.1.2.6 Next steps

Current MPTCP specification [49] is experimental. There is an ongoing effort to move it to Standards track. We believe that the work on MPTCP security should follow two treads:

- The work on improving MPTCP security enough to become a Standard Track document.
- The work on analyzing possible additional security enhancements to provide a more secure version of MPTCP.

We will expand on these two next.

2.1.2.6.1 MPTCP security for a Standard Track specification. We believe that in order for MPTCP to progress to Standard Track, the ADD-ADDR attack must be addressed. We believe that the solution that should be adopted in order to deal with this attack is to include an HMAC to the ADD ADDR message (with the address being added used as input to the HMAC, as well as the key). This would make the ADD ADDR message as secure as the JOIN message. In addition, this implies that if we implement a more secure way to create the key used in the MPTCP connection, it can be used to improve the security of both the JOIN and the ADD ADDR message automatically (since both use the same key in the HMAC).

2.1.2.6.2 Security enhancements for MPTCP. We also believe that is worthwhile exploring alternatives to secure MPTCP. As we identified earlier, the problem is securing JOIN messages is fundamentally incompatible with NAT support, so it is likely that a solution to this problem involves the protection of the data itself. Exploring the integration of MPTCP and approaches like TCPCrypt seems a promising venue.

2.1.3 New MPTCP use cases

Currently, the main use case for MPTCP is to pool bandwidth by splitting the traffic of a MPTCP connection over the different interfaces locally available in the host. In this section we propose new use cases for MPTCP. First, we present three new MPTCP use cases based on the observation to that MPTCP is a perfect fit for explicitly addressing middleboxes along the different paths. We propose three novel use cases for MPTCP, namely connection/waypoint redirection, connection endpoint migration and wide area virtual machine migration related to the middlebox integration scenario.

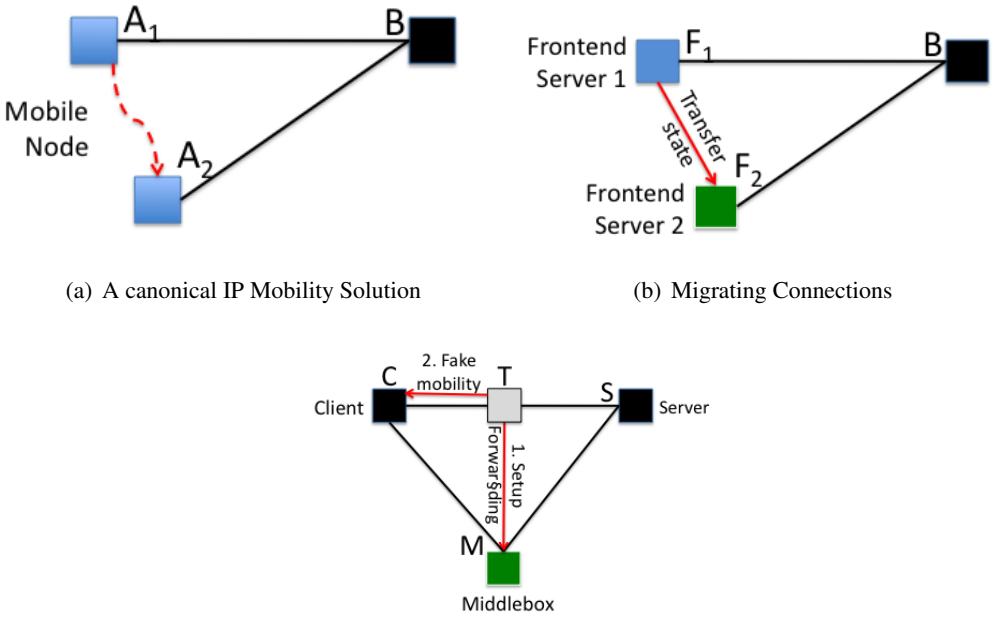
Then, we present a new MPTCP use case for the mobile/wireless scenario. We propose to use MPTCP to opportunistically share cellular links between mobile devices, reducing the energy consumption and optimizing the use of cellular Internet access links.

2.1.3.1 Integrating middleboxes with MPTCP

Middleboxes form an integral part of the Internet today : operators, major content providers and even certain applications use middleboxes to bypass the shortcomings of the Internet architecture including security, load balancing, in-bound connectivity behind NATs, and so forth. However, middleboxes also make the Internet brittle, difficult to debug and evolve. There is wealth of research to overcome these issues by properly including middleboxes in the Internet architecture, e.g. [145, 58]. However, these solutions depend on changes to the IP layer or upgrades to all applications, none of which seem feasible in the near future.

We ask the pragmatic question of how to incorporate middleboxes into the Internet architecture in a deployable way. We survey existing proposals and find that deployability severely restricts possible changes. That is why we restrict ourselves to a) making operator-deployed middleboxes explicit in the data path (as opposed to transparent), and b) making provider middleboxes' flow handling more flexible.

To support these goals, we leverage the Multipath TCP protocol. To achieve multipath operation, MPTCP implements two key constructs: a per-host opaque connection identifier (not related to IP addresses), and a way to add new addresses to existing connections. These two constructs enable **connection acrobatics**:



(a) A canonical IP Mobility Solution

(b) Migrating Connections

(c) Connection redirection

Figure 2.11: Connection acrobatics

the ability to explicitly redirect connections via a waypoint, and the ability to migrate the endpoint of a connection. We show that connection acrobatics are *sufficient* to achieve both our goals. Further, we propose the "sticky bit", an optimization to MPTCP that can reduce the redirection overheads for short connections.

Flexibility has a cost: attackers can abuse these same mechanisms to easily hijack connections and move them off-path, especially when the sticky-bit is used. We discuss these vulnerabilities and potential solutions to mitigate them. Finally we explore and implement a number of useful scenarios that are enabled by connection acrobatics in section 2.1.3.1.5. A short paper on connection acrobatics has been presented at the HotMiddlebox workshop in December 2013.

2.1.3.1.1 Problem Statement Most of today's connections pass via two middlebox domains. First, "eyeball" networks use routing (e.g. OSPF) and tunneling (MPLS) techniques to steer customer traffic to middleboxes such as firewalls, NATs and performance enhancing proxies. Unfortunately, routing techniques are prone to misconfiguration and do not scale well. Additionally, such middleboxes are transparent to the end-hosts which makes it difficult to debug connectivity problems and reason about end-to-end behavior of protocol extensions [66].

Content-providers use DNS to attract customer connections to front-end servers that terminate the connections, authenticate the customers and load balance each request to the best server. This setup is inflexible: the front-end does not need to be on the data path after client authentication; a better front-end may exist if the client moves.

The research literature has proposed many ways to incorporate middleboxes into the Internet architecture including Delegation Oriented Architecture (DOA) [145] and NUTSS[58]. The key findings are that:

(i) Explicit connection signaling is needed to negotiate connectivity through firewalls; this ensures policy compliance and network evolvability as it decouples the control traffic needed to setup the connection from the data traffic (which could use a different protocol).

(ii) Middleboxes should only process traffic explicitly addressed to them. Having explicit addressing would reduce routing complexity and makes it easier to debug the network.

Despite their appeal, none of the existing solutions has been deployed. DOA requires changes to IP, and NUTSS requires changes to the socket API which implies all apps must be changed to take advantage.

Our focus is to implement these principles in a deployable system which has to meet the following two goals: a) applications should not need changing to take advantage of it, and b) it has to be compatible with the current network.

A fundamental problem of the current Internet is that the network does not know with certainty what application traffic it is forwarding. Ideally, port numbers should identify the end-host application but the trend of consolidating all traffic on few ports (in particular port 80) reduces accuracy drastically. However such knowledge is needed to decide whether traffic should be allowed, or to optimize the traffic, for instance by routing it via application-specific middleboxes.

Ideally, applications should explicitly signal to middleboxes what they want to do, but this is impossible without changing applications. Instead, so we have to make due with the status quo: networks have begun to rely on deep packet inspection boxes that use various heuristics to categorize traffic. In many cases, it is impossible to know during the three-way handshake what app a connection is serving: only after the Deep Packet Inspection (DPI) box sees enough traffic it can accurately identify the application and steer the traffic to the appropriate middlebox. Today redirection is done transparently, without the end-host knowing, inflating the round-trip time and creating mysterious failure modes. Instead, we would like the connection to be explicitly redirected via the middlebox, with the end-hosts' knowledge.

Pragmatically, we need a way to **move the middle of a connection** via a specified waypoint (i.e. a middlebox). This scenario is shown in Figure 2.11(c), where the transparent box T redirects the established connection C-S via M, effectively transforming it in two connections: C-M and M-S. To implement the redirection, box T must change the data plane by re-routing the flow, and will also need to control middlebox M to instruct it to proxy the new flow. Note that the difficulty is in changing the data plane, as the control plane operations in this case are rather simple: as long as M trusts T, SNMP, OpenFlow or any other network configuration protocol can be used to configure the forwarding state at M. Once M sees the traffic, it should be possible to reroute it via another waypoint if it wishes, and so on.

On the server side, DNS will direct the client's traffic to a nearby datacenter where a front-end proxy that will terminate the mobile connection (and maybe TLS) and authenticate the client before passing the request to the real server. There are two issues with this setup: first, the front-end does not need be on the data path after the authentication, and keeping it there increases its load and general complexity. Second when a front-end

needs to be taken offline (e.g. for maintenance) all the connections it serves will simply die.

We observe that on the server side the limitations come from TCP's reliance on a fixed set of addresses: a connection cannot be efficiently migrated to a different front-end unless in the same layer 2 domain. Thus, a solution must **move the endpoint of a TCP connection** to allow better load balancing and traffic optimization.

2.1.3.1.2 Solution Space We have found that explicit signaling cannot be obtained in a deployable way—it requires application changes. Luckily, the goal of making middleboxes explicit should be achievable as long as we can move both the middle and the "ends" of a connection.

We start by observing that any mobility solution can also easily support connection endpoint migration, as long as it is implemented at layer 4 or below. This is quite surprising, as mobility solutions were only built to cope with the devices changing their point of attachment to the network.

To understand why mobility can help, consider a canonical mobility solution as shown in Figure 2.11(a) that offers support for a mobile node to continue connectivity despite changing its network attachment point from A_1 to A_2 in the figure. Endpoint migration is easy to implement using this mechanism as shown in Figure 2.11(b):

(i) Transfer state from the origin server (front-end 1 in the figure) to the destination server (front-end 2) including TCP connection state (sequence numbers, IP addresses, ports, send and receive buffers, etc) and mobility mechanism state (e.g. keys used for the association with the remote node).

(ii) Pretend that the Front-end Server 1 has changed attachment point from A_1 to A_2, by using the mobility mechanism. The exact details depend on the mobility mechanism used.

Moving the middle of a connection (as shown in Figure 2.11(c)) can also be implemented on top of mobility. The middlebox must trigger two mobility events: to the client, it will appear that the server has moved to middlebox M, while to the server the client will appear to move to M. In effect, T has to fake two mobility events; of course, the protocol mechanisms depend on the mobility solution used, but it should be feasible to do this with any mobility mechanism.

The one caveat for redirection is that the middlebox T may not know the keys used by the client and the server to secure IP mobility. In this case, T has to ask for help from one endpoint, asking it to fake a mobility event (e.g. client moves to M). We discuss a few concrete security issues in more detail in section 2.1.3.1.4.

2.1.3.1.2.1 Choosing a mobility mechanism Mobile IP [107], Locator-ID Separation Protocol (LISP) [42], Shim6 [95] or Host Identity Protocol (HIP) [89] are only a few of the technologies that have been devised to support mobility, so there is a wealth to choose from. More recently, Serval [97] proposes to use service names instead of IP addresses in a bid to optimize the Internet for serving content; Serval also allows connection migration and mobility by design. All these solutions function (mostly) transparently to the transport layer. The upshot is they support redirections for all traffic, not just TCP; the bad part is that,

at least for traditional mobility mechanisms, redirections are done in bulk, and not for specific connections, but this should be possible to fix. Even worse, IP-level solutions are oblivious to the transport protocol which can cause TCP performance to drop significantly when mobility events are frequent [124]. Crucially, none of these solutions is deployed—or showing signs of imminent deployment.

There are many proposals that offer mobility at transport level, including SCTP, Migratory-TCP [46] and TCP Migrate [131]. All of them support mobility by adding a per-connection unique identifier independent of the endpoints' IP addresses; this identifier allows the connection to resume when the address changes. SCTP is a mature and feature-rich protocol, yet it has not seen deployment because middleboxes (e.g. NATs) block it. The TCP-based solutions should be deployable, yet they haven't seen adoption partly because of lack of maturity, and partly because of wrong timing—when they were proposed, mobility problems were infrequent in practice. Surprisingly, the newly standardized and deployed MPTCP protocol seems the best bet for an easily deployable solution.

2.1.3.1.3 Integrating middleboxes with MPTCP Multipath TCP allows the efficient utilization of multiple network paths within the same transport connection in a way that is transparent to applications and the network [117]. Because of this requirement, MPTCP allows endpoints to add new addresses to existing connections: every MPTCP connection is given a unique identifier by each endpoint upon creation, which is a hash of a key chosen by that endpoint. The keys are carried in the initial handshake in new TCP-options.

When a new subflow is being added to an existing connection, the MP_JOIN option in the SYN informs the remote end that this is a part of an existing connection, rather then a new one; the option is cryptographically signed using a concatenation of the connection keys. The aim of these mechanisms it to ensure that only the endpoints can add new subflows; however boxes that are on-path for the initial handshake know the keys and can add subflows at will.

The MPTCP standard also allows endpoints to advertise addresses by adding an ADD_ADDR option to any segment; the receiving end would then start a new subflow using the advertised address. Note that ADD_ADDR options are not protected by the connection key: thus anyone on-path can add an address, not just the endpoints. This can result in a security breach that we address in Section 2.1.3.1.4.

Multipath TCP's address management mechanisms are sufficient to implement the needed redirection mechanisms. For redirection, the ADD_ADDR functionality can be used to redirect traffic as follows. Say an MPTCP connection is setup between C and S (see Figure 2.11(c)). T sets up a forwarding rule at M instructing it to proxy all traffic it receives from C. Then, T sends an ADD_ADDR message to C advertising M's address. As a result, C will send a SYN+MP_JOIN message to initiate the three way handshake. M receives the SYN+MP_JOIN message, it changes the source address to M and the destination address to S and forwards the message to S. An analogous processing is applied to subsequent packets of the three way handshake. The result is that Both C and S have a new subflow with M, which acts as an explicit middlebox for the MPTCP connection. T can now close the C-S subflow, effectively forcing the endpoints to use only

the path via M. Once M sees the traffic, it can direct the traffic to S or D via another proxy by simply using ADD_ADDR and setting up the appropriate proxy rules. We have implemented this type of redirection and tested it in practice. The experimental results are discussed in Section 2.1.3.1.5.

The IOS implementation of MPTCP does not implement ADD_ADDR as it perceives it as a security threat [37]. However, redirection with MPTCP is still possible by having the middlebox explicitly initiate two subflows to the endpoints, as follows: M learns the connection keys from T and then sends a SYN+MP_JOIN message to S, mimicking standard subflow creation with MPTCP. S will reply, finalizing the handshake. Similarly, M has to setup a subflow to C by sending a SYN+MP_JOIN. As long as C is not behind a NAT, this type of redirection works equally well.

Connection migration. Connection migration is easy to implement by migrating the MPTCP connection state and opening a new subflow from the new location. However migrating the related application logic is quite difficult. Here, we distinguish two cases: in the first case, the application itself is migrated to the new machine, either by process migration or virtual machine migration, and it just keeps running. In the second case, the connection alone is migrated from one instance of an application to another. If the connection is moved mid-stream, synchronizing the application state obviously requires non-trivial application updates in the general case, and service continuations could be used for this purpose [133]. However, such application changes defeat our deployability goal.

We observe that there are scenarios where connection migration does not require application changes: one example is migrating just after connection startup and before the application receives notification that the connection is ready (i.e. before connect or accept returns). In this case, it is safe to migrate as the application hasn't had the chance to change its state. This functionality can be useful to implement anycast, for instance: the initial handshake is terminated by the load balancer, which then selects a target server and migrates the whole connection there. We have implemented this functionality and show experiments in Section 2.1.3.1.5.

Another example targets applications that use TLS with the TLS API (*ssl_connect*, *ssl_accept*): it is safe to migrate the connection after the SSL handshake completes but before any data is sent by the application. This use-case requires changes to the TLS library, though, so that the connection keys are migrated too. It can be used to offload the expensive TLS handshake (because of public-key operations) to specialized boxes, and then move the connection to the actual servers.

Optimizations. Redirection with MPTCP requires setting up an additional subflow, which incurs a non-negligible overhead and also delays application traffic. In practice, once a redirection has been made, it is quite likely that traffic to the same server and port will be redirected in the same way in the future—for instance, a large number of HTTP connections are made to the same server in a short period of time, so they should be redirected in the same way.

We can exploit such "connection-locality" by adding a *sticky bit* to the ADD_ADDR option. Upon reception of such an option, the receiver will save the address in its local connection cache and use it for future

connections: in effect the client remembers to directly use the “optimal” connection. This optimization has security implications that we discuss next.

2.1.3.1.4 Security Analysis Connection acrobatics allow redirecting MPTCP connections to middleboxes located anywhere in the Internet, so their implications need to be carefully considered from a security perspective. The design of MPTCP takes into account a wide range of security threats [10]. The analysis from section 2.1.2 identified a few remaining potential threats as well as the required countermeasures to prevent them.

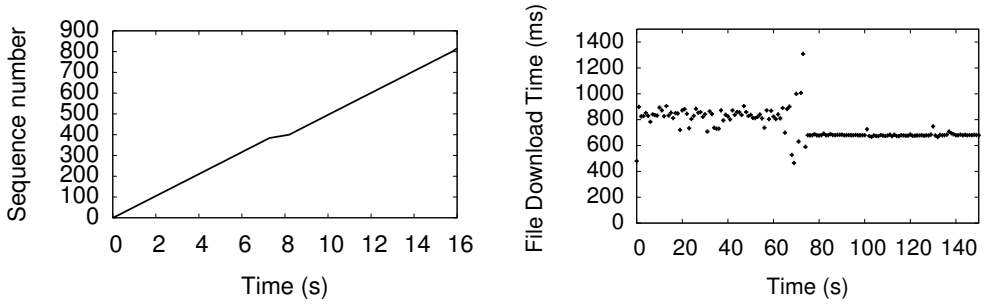
The design goal for MPTCP security is simple: an ongoing connection can only be redirected by the connection endpoints or other parties explicitly authorized by the connection endpoints to do so. This is achieved by securing the messages that add a new address in the ongoing connection using cryptographic material generated at the beginning of the connection lifetime.

In the current MPTCP specification, the cryptographic material is exchanged in clear during the connection set up. All forthcoming control messages that are used to divert traffic are protected with keys generated from this cryptographic material The result is that MPTCP is vulnerable to attackers that are located along the path during the connection establishment phase. This is similar to the protection SCTP has and it is acceptable as long as only one MPTCP connection is at stake.

The notable shortcoming in the current MPTCP specification identified in [11] is that the ADD_ADDR message is not protected and can be used by an off-path attacker to blindly divert the communication to an arbitrary address in the Internet (basically using the same approach we propose for a middlebox to redirect the traffic). The solution to this security breach is to include an HMAC protection using the security token negotiated during the MPTCP connection establishment phase. This would enable the safe operation of the techniques we propose. The implication is that the middlebox must know the security token in order to generate an ADD_ADDR message, but this can be conveyed using the control protocol used to create the forwarding rule in the middlebox as described earlier.

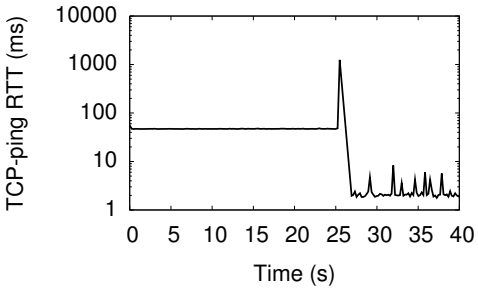
However, when we consider the ”sticky” bit as described earlier, it is not only the ongoing connection that is protected by the initial cryptographic material, but also all future MPTCP connections. Thus, the use of the ”sticky” bit requires stronger security. This can be achieved by generating the keys used to secure the redirection messages in a more secure way. In particular, it is possible to use SSL keys to do that as described in [104]. Indeed, many data-transfers nowadays rely on TLS/SSL to secure the connection. TLS/SSL negotiate between the two end-hosts a key that is not sent in clear over the network and through server-certificates it is not even possible for a Man-in-the-Middle attacker to interfere with the connection. MPTCP could use the key from TLS to authenticate new subflows which would greatly increase the security of MPTCP as the keys are no longer sent in clear. A proposal to reuse the TLS key to secure MPTCP is described in [104].

In summary, as long as the ”sticky” bit is not used, security can be achieved by HMAC-ing the ADD_ADDR options. The “sticky” bit is only safe to use when MPTCP’s keys are truly secret (e.g. reuse TLS keys).



(a) End-to-end traffic is not affected as the connection is redirected from T to M

(b) Using Connection Migration for Load Balancing



(c) Live Migrating a Virtual Machine from Romania to Germany

2.1.3.1.5 Implementation Using the MPTCP kernel code as basis, we have implemented connection acrobatics. The implementation required very few changes to the MPTCP stack itself, and no changes whatsoever to the network boxes or to the application code, giving us confidence that the changes should be easily deployable. In this section we explore the usefulness of acrobatics by experimenting with three key use-cases and discussing their implications for the Internet at large.

2.1.3.1.5.1 Connection Redirection To move the "middle" of a connection, we use a setup where the client and the server have an established long-running TCP connection. Explicit redirection is impossible with vanilla TCP [4], but it is trivial with MPTCP and requires no protocol or kernel changes. Our transparent middlebox T (see figure 2.11(c)) runs on a plain Linux box, where our simple user-space program monitors interfaces using *pcap* and records connection details (sequence numbers, IP addresses, ports). To redirect the connection via middlebox M, the program takes these steps:

- T sends connection details to M, which installs forwarding rules to handle the traffic.
- Once M acknowledges the rules have been setup, T injects a TCP keep-alive for the client, carrying an ADD_ADDR option advertising the address of the second middlebox.
- The client will open a new subflow to M which uses simple netfilter rules to proxy the packet. This ensures M will be on the path of the new subflow.
- T closes the initial subflow by injecting TCP RST segments in both directions, effectively forcing the client and the server to use the other subflow.

[4] An operator can always use routing or tunneling for redirection, but this is both limited in scope and inefficient.

Using this setup, an operator could dynamically redirect traffic to different middleboxes as required by policy. For instance, traffic containing certain keywords could be redirected to a full-blown intrusion detection system for more detailed checks.

In our experiment, T scans the TCP payload for the word "secure", and redirects the connection when a match is found. Figure 2.12(a) plots the sequence number progression at the receiver, showing that redirection has a negligible effect of performance. Redirection is quick: it takes two round trip times between the middleboxes plus the time needed for creating the subflow; in this example the total redirection time is around 200ms.

Connection redirection enables explicit middleboxes and allows arbitrary middlebox chaining, but it can do more than that. It also enables a few interesting features that are difficult to implement today:

- **Destination Routing Control:** load-balancing for inbound traffic is notoriously difficult to implement in the Internet, leading to hacks such as BGP path poisoning [75]. With redirection, **any box** receiving traffic can choose to redirect it via a waypoint. In figure 2.11(c), M could choose to further redirect the traffic if it wishes, by sending ADD_ADDR itself.

- **Load balancing.** Using redirection, operators can load balance their networks in a fine grained manner, by targeting only long-running connections and redirecting them to proxies in different parts of the network. Alternatives include Equal Cost Multipath Routing (not good when routes are not equal cost) or MPLS-TE, which works on traffic aggregates.

- **Transition to IPv6.** IPv6 deployment is sluggish despite widespread OS support and major network upgrades. The problem is that it's difficult to know whether the IPv6 path works fine. Using redirection, an operator could advertise IPv6 addresses in its own network to make clients send traffic over IPv6. If the server supports IPv6, the connection can continue on IPv6 until the destination; otherwise the hop between M and the server could use IPv4. The advantage of this solution over Happy Eyeballs [151] is that traffic can be migrated back if the IPv6 network is overloaded.

2.1.3.1.5.2 Migrating connection endpoints To support endpoint migration (Fig. 2.11(b)) we have changed the MPTCP stack, but not the protocol itself. The client connects to an MPTCP-enabled server F_1 which completes the three-way handshake but does not pass the connection to the application. Instead, F_1 transfers the crypto material—the 2 keys exchanged at connection setup—to F_2. At this point F_2 has enough information to accept a new subflow from the client; once the new subflow is setup it can pass the new socket to the waiting process, by returning from the accept syscall. Most importantly, our implementation does not require any application changes. Clients do a normal `connect(2)` to the first server which decides to redirect the connection without involving the application; in turn, the second server uses a simple `accept(2)` as if the client had connected directly.

This simple setup could effectively be used as a low-cost method of coping with flash-crowds. Consider a small web provider that suddenly becomes very popular. Once its server detects it is under load, a copy of

the server could be spawned by renting a VM from a public cloud provider such as EC2. To balance the load across the two servers, the traditional approach is to use DNS-based redirection. Unfortunately, it takes a long time for the new DNS records to propagate because of caching in the DNS hierarchy. The web provider could choose very small TTL to limit the amount of caching, but this is costly as it would unnecessarily and permanently increase the load on the provider's DNS server.

Instead, the provider can use acrobatics to perform short term load balancing: with only a few CPU cycles per connection it can complete the three-way handshake; then it will immediately redirect a fraction of the incoming connections to the other server.

To emulate such a scenario, we ran a simple experiment where an Apache server with a gigabit link serves the same 25MB file to two customers that download it over and over again. The plot in figure 2.12(b) shows the file download times as time progresses. At around 70s, the server decides to load balance half of its requests to another similar server by migrating connections after the three-way handshake is finalized. The graphs shows that load balancing reduces both the average download time as well as the variance.

Connection migration also enables a few features missing in the Internet architecture today, including:

- **TLS Handshake Offloading.** A more secure version of endpoint migration is to use TLS keys to secure MPTCP, and to migrate a connection after the TLS handshake has finished. This would allow offloading the expensive TLS handshake to specialized boxes, and the connection would be then redirected to the final servers.

- **Off by default!** By using TLS Handshake Offloading, servers could become effectively *off by default* [13]: only successfully authenticated clients would be allowed to contact the destination server. Once a new connection is allowed and thus migrated, the destination firewall will be configured to accept SYN packets with the correct tokens, filtering out all other traffic; the token is similar to a capability in [13]. The TLS infrastructure needs be distributed to ensure it can withstand large scale attacks; however, such a deployment could be from a third party (e.g. Akamai).

- **Anycast.** Load-balancing proxies can move the connection to the chosen server, removing themselves out of the data path after the initial handshake. This allows a (two-stage) anycast that is friendly to TCP; the alternative of using BGP anycast is brittle as TCP connections will break when routing tables change.

2.1.3.1.5.3 Wide Area Virtual Machine Migration Virtual machines help increase server utilization while isolating potentially distrusting users. They are one of the key enabling technologies in public clouds, and we are witnessing a shift to also have on-demand general purpose processing in operator networks, in the form of micro-datacenters. Content providers could rent VMs running in micro-datacenters to run frontend proxies as close to the customer as possible.

Virtual machine migration is widely used inside datacenters because it allows load balancing and planned

hardware maintenance operations. Migration is great as long as the VM can keep its IP address: otherwise ongoing TCP connections will break. That is why today migration is restricted to the same L2 domain. It would be nice if providers could seamlessly migrate VMs between micro-datacenters in response to changing user behavior.

MPTCP allows wide-area VM migration out-of-the-box: we have successfully migrated an MPTCP virtual machine in the wide-area without any changes to Xen or to the MPTCP stack; we only needed to implement a simple user-space program that installs a Xenstore watch and monitors a key specific to the current Hypervisor. We monitor the Domain ID, which (typically) changes when migrating to a different hypervisor. When the machine is migrated, the monitored key changes and the program triggers an address renewal (a DHCP request). Upon receiving a new address, the MPTCP stack automatically opens a new subflow for each of the ongoing MPTCP connections.

Fig. 2.12(c) shows the application level round-trip as measured by a client running in Germany that sends periodic requests over a TCP connection to a virtual machine in Romania. 25 seconds into the experiment, the virtual machine gets migrated to Germany, and the RTT drops from 50ms to around 4ms. The perturbation induced by the migration lasts for a few seconds, increasing RTTs to 1.2s.

Wide area VM migration enables a number of interesting use-cases including the ability to take down an entire datacenter, and the ability to shift processing to follow cheap electricity in diurnal patterns (i.e. follow-the-sun load balancing). Crucially, it allows content-providers to migrate their middleboxes (i.e. front-end servers) when needed without breaking connectivity.

2.1.3.1.6 Conclusions

Connection acrobatics represent a pragmatic step towards embedding middleboxes into the Internet architecture, be they transparent operator-deployed machines or explicit proxies deployed by the content provider. To ensure deployability, we have limited ourselves to data-path changes—in particular, what can we do to make middleboxes explicit and more flexible? We find that, surprisingly, any mobility solution can be used to solve our problem, but we choose to focus on Multipath TCP.

MPTCP offers a degree of flexibility that is badly needed in the Internet. Our implementation and experiments show connection redirection, connection-migration and virtual machine migration are efficient and perform well in practice. These tools can help fix other problems of the Internet today including the difficulty of load balancing traffic and can help protect against DDoS attacks.

2.1.3.2 Improving cellular connectivity with the Mobile Kibbutz

Mobile devices have become more numerous than the traditional desktop computers, and much of their success is due to the "always-connected" paradigm they enable: users are constantly in touch with the latest news, receive email, navigate and chat while on the move.

Connectivity is so important that mobile devices support multiple wireless technologies including cellular (3G, LTE), WiFi (802.11), and Bluetooth. These technologies offer different tradeoffs with respect to distance, RTT, and per-bit energy consumption. WiFi links offer high bandwidth, small RTT and small -per-bit

energy consumption, but coverage is patchy, with large areas uncovered. Cellular technologies have ubiquitous coverage and offer a mobile "always-connected" experience; that is why they are the default choice for connectivity. On the downside, they are both energy-hungry and have huge RTTs after periods of inactivity. Bluetooth connectivity on the other hand is geared towards personal networking, with short distance, low power and low bandwidth connectivity.

In cellular, high RTTs occur due to signaling that implements centralized control of capacity sharing, prevalent in cellular networks. Centralized capacity allocation is here to stay as it enables high utilization of resources and strong fairness guarantees which are not available in distributed access control such as CSMA. Finally, excessive energy consumption is a consequence of trying to minimize the amount of signaling and its implicit costs. In fact, the state of the art in cellular networks strikes a balance between signaling and energy consumption by using provider-configured or dynamic [33] inactivity timers to govern mobile radio state transitions. So far, timer-based solutions make strong assumptions about traffic patterns and has cases where it incurs excessive delay and/or energy consumption.

An opportunistic solution to mitigate the shortcomings of cellular connections is proposed. This solution does not apply to individual devices, but rather to groups of mobile devices, and it is called the Mobile Kibbutz. The basic idea is for nearby mobile devices to share their cellular connections with each other via WiFi or Bluetooth links. There are ample opportunities for users to share connections [86], and ample incentives to make ones data plan more 'liquid' [70]. 3G link characteristics are such that periodic traffic consumes disproportionate energy, and bursty web-like traffic has low responsiveness. The basic idea of the Kibbutz is for users to take turns in using each others 3G connection such that battery consumption and responsiveness are both improved, all while ensuring fairness, accountability and privacy of participants.

Implementing the Kibbutz faces a number of issues, among which the need to steer mobile devices' traffic to one or a subset of dynamically changing cellular links stands out. If a TCP connection is long-lived, migrating it from one link to the other is not possible, and the connection will break. To avoid these problems the newly standardized MPTCP protocol [50] that allows seamless migration of connections across links is used, as well as multiple links for a single connection.

The Kibbutz implements a tit-for-tat policy that ensures short time fairness for energy consumption. The aim is not to impose short or long term bandwidth fairness, but rather to allow users to consume as much as they want. A Kibbutz service is proposed to securely measures all the traffic sent by each user via its neighbors and on behalf of its neighbors; these measurements can be used to implement any billing policy (e.g. fair share, billing). This section is based on a paper submitted to the ACM CoNEXT 2013 conference.

2.1.3.2.1 Mobile networking today

Cellular links such as 3G and LTE are widely available geographically, being the default interface for devices. Cellular power consumption depends on the state of the connected link: when the link is idle, its power consumption is close to zero. When the device has to transmit or receive a single typically-sized packet (e.g.

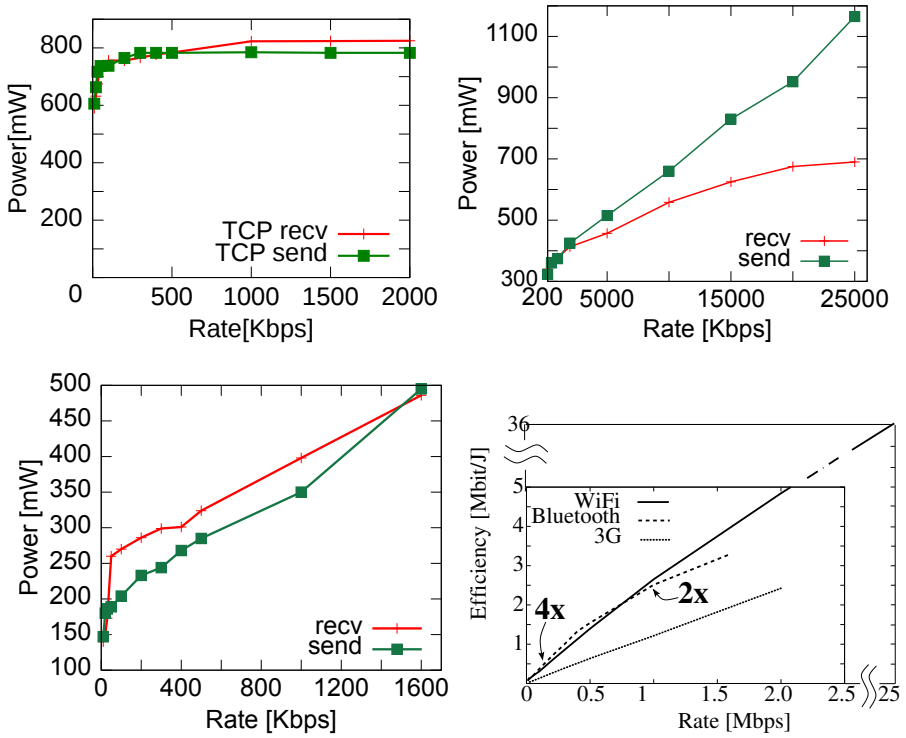


Figure 2.12: Power consumption of a Galaxy Nexus phone generating traffic at various rates over different wireless interfaces: a) 3G power consumption with TCP traffic b) WiFi power consumption (UDP traffic) c) Bluetooth power consumption (UDP) d) Efficiency compared: at 150Kbps 3G is 4 times worse than WiFi or BT.

1KB), it will transition into a high power state. The transition takes seconds and is expensive energetically because of signaling with the base station. To amortize its cost across many packets, the solution used in practice is for the mobile device to remain in the high-power state for a predefined amount of time after the last packet has been sent (5-10 seconds, depending on the operator). When very little traffic is sent, this tail wastes precious energy, draining batteries.

This behavior is widely studied in recent literature [12, 125, 33, 69], and also visible in the Kibbutz experiments (see Figure 2.19 left, labeled 'Stand alone'). Any relevant transfers are performed in the high power DCH[5] state - in Kibbutz tests, the transition to this state takes about 2.3s up, and 7.3s down for one 3G operator. The sum of these values seem to vary slightly with time of the day, but stays roughly in the 5s-10s region.

LTE networks behave similarly. There are quantitative differences, though: total energy consumption is higher, on the order of 1-4W, while the setup time is a bit smaller, between 0.5s-1s in practice [69, 33].

This behavior has two critical implications for interactive traffic: first, after a silence, the mobile will have to wait the 2.3s setup time before sending *any* traffic, and second, waste about 3J of energy to wind down. In addition, any amount of traffic with a period of less than 5-7s will keep the radio in the high power state, regardless of the actual transfer rate.

In fact, power consumption tends to have a flat response to increasing traffic. Beyond a certain inter-arrival rate, the 3G interface remains in the high power state, and variations in consumption depend only on the cost

[5]FACH=Forward access channel (medium power, common channel); DCH=Dedicated channel (high power)

of moving data through the stack, as confirmed by measurements in Figure 2.12(d).

WiFi links offer better throughput than their cellular counterparts, but their coverage is patchy. Most phones use the 802.11 power save mode permanently in order to power down the interface when little traffic is to be transferred, making WiFi power consumption proportional to the traffic rate. In Figure 2.12(e), the power consumption for different rates is plotted, and it is mostly linear for rates above 200Kbps. Part of the linearity of the curve is given by the cost of the software stack, as was recently shown in [51].

Bluetooth links (3.0) are only available at a few meters and provides below 2Mbps in the reported tests. On the other hand, it presents a power consumption curve that is linear with the throughput achieved (Figure 2.12(f)).

To emphasize the difference in power consumption patterns between the three radios, in Figure 2.12(g) the measured data is recasted to show the efficiency measured in Mbit/J. The efficiency does increase for all three technologies due to amortization. For 3G the tail end gets used better with increasing traffic, when more bits are handled with the same cost. For Bluetooth and WiFi, powering the circuitry takes proportionally more than the software stack - in Figures 2.12(e) and 2.12(f) the line doesn't actually pass through 0. Note that at lower rates, the efficiency of 3G is much worse, due to the long inactivity timer in the DCH and FACH states. Beyond 750Kbps WiFi is more efficient than Bluetooth.

Mobile Applications fall in two broad classes. Background apps synchronize periodically with servers, have low rates and waste energy because of the tail timer. Optimizing energy consumption for such apps has been an active research area, with quite a few solutions proposed [112, 111, 12, 125, 33].

Interactive applications can be characterized in other three classes: streaming (VOIP, radio, online gaming, video streaming), web browsing and app downloads.

Streaming apps utilize cellular links inefficiently: they consume little bandwidth (tens of Kbps) and have large packet inter-arrival times. Web browsing works in a user-driven closed-loop: a burst of traffic downloading a page triggers a costly energy tail, followed by a relatively short period of silence that allows the radio to go IDLE. When the next burst arrives, it has to wait a few seconds for a state transition. Finally, app downloads utilize the full speed of the cellular interface for brief periods. Here, higher download speeds are preferable, but users are limited by the cellular capacity.

Running these apps over cellular links leads to high energy costs and poor overall performance. Ideally, the aim is to run all these applications over WiFi-like links that offer energy proportionality and high speeds, but such links are not available all the time.

2.1.3.2.2 The Mobile Kibbutz

The key insights from the above measurements are:

- Cellular links offer good interactive experience only when in the high-power state. While in this state, they draw the same power regardless of the traffic load.

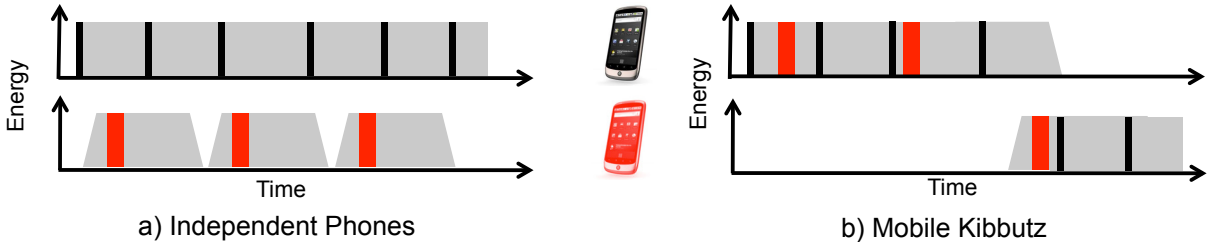


Figure 2.13: The mobile kibbutz consolidates traffic onto a single link, reducing energy consumption and delay at the same time. Energy consumed by actual traffic shown as red and black bars, and wasted energy is shown in gray.

- Many mobile applications such as web radio, or web browsing under-utilize cellular links most of the time.
- WiFi and Bluetooth links offer power-proportionality and low delays.

The Kibbutz goal is to achieve low energy consumption, small RTTs, and fair privacy aware sharing of cellular links, all while maintaining the TCP socket interface to existing applications. To the best of our knowledge, there are no solutions that achieve this goal today.

Work on optimizing energy consumption for cellular links has tried to either *adaptively reduce the tail timer* or *to cluster packets in time by delaying some of them*. The first solution works reasonably well for background applications, but it is not applicable to applications that send at low data rates (e.g. radio, streaming). The second solution requires application changes, and only applies to delay-tolerant apps.

The proposed solution is based on the observation that, in most cases, mobile users roam in dense areas where many other mobile users are around (public spaces, transport, office, home). The idea is to join nearby mobile devices into a Mobile Kibbutz, which allows devices to use each other's cellular connection by forwarding traffic along cheaper WiFi or Bluetooth links. By joining the kibbutz, devices can dynamically share their cellular connections with the rest of their group for an advertised period. While the advertisement is valid, neighboring devices have a choice between using their own link or the shared link for sending or receiving traffic.

The kibbutz allows devices to consolidate their traffic onto a single cellular link, reducing both energy consumption and delay across all devices. Consolidation brings cellular links closer to the their optimal operating point: one link will be highly loaded while others will be idle.

The concept is presented in Figure 2.13. The black user is listening to Internet radio, while the red user is browsing the web. In isolation, both users keep their cellular links busy most of the time (Figure 2.13.a), despite the low data rate. In addition, the red user experiences high delays when accessing every single page because his link must transition from the IDLE to the DCH state. By consolidating traffic onto a single link, the other link can be left IDLE all the time; this reduces the total energy consumption to approximately half. Additionally, the red user's delay decreases because state transitions are unnecessary most of the time.

The mobile devices use MPTCP to dynamically steer traffic over one or a subset of available links. In

effect, the kibbutz *pools together cellular links*, enabling many optimization avenues. The most simple optimization is bandwidth: a device can use a neighbor's bandwidth when the latter is not using it, thus increasing download speeds and user experience. In the following sub-sections the operation of the Mobile Kibbutz is described in detail.

2.1.3.2.2.1 Basic Operation

To use the Mobile Kibbutz the user has to first connect to other nearby devices. There are a number of technologies that can be used for this purpose:

- **WiFi Ad-Hoc** is the favorite technology because it allows true peer-to-peer operation and high speeds; unfortunately support for Ad-Hoc is being phased out of mobile WiFi drivers.
- **Bluetooth** is the second best option: it is energy efficient but it only supports point-to-point connections, so it will be difficult to create large groups of devices because of the sheer number of connections required. Additionally, Bluetooth has relatively low speeds and pairing requires user input: users will have to explicitly agree when the device wants to form a kibbutz with an unknown device.
- **WiFi hotspot** mode (or WiFi direct) is when one device acts as the access point, and the other as client. This mode is widely supported and offers high speeds. On the downside the setup is asymmetric as the access point will consume more power (200mW on the Galaxy Nexus).

Advertisements. Once the device is connected to a kibbutz, it can share its cellular connection with the other devices. One simple option is to always share the connection, but this may quickly drain the device's battery. A better option is to share the link only when it is already in the high power state: this way the additional energy consumed to forward the neighbor's traffic is very small and the traffic will experience low delays. The default sharing algorithm is for device A to advertise its link when one of its own packets leaves the interface and the interface is not already shared. A's advertisement includes an expiration time t that tells peers when to stop forwarding traffic via A.

The choice of t affects the performance of the kibbutz; smaller t leads to more signaling required for advertisements, and it could also lead to frequent switches of traffic between devices, thus negating the energy savings the kibbutz can bring. That is why Mobile Kibbutz uses an advertisement period equal to the tail timer of the mobile operator. This value is the largest that guarantees the link will always be in the high power state when used by peers. Sharing an IDLE link would be bad: it costs a state transition and provides increased delay.

The default advertisement algorithm can lead to unfairness: in many cases the whole kibbutz will end up using a single device's link. Additionally, it is very easy for a device to leech on other devices' resources, undermining the kibbutz. To incentivize devices to share their links a simple tit-for-tat algorithm is adopted: each device keeps a counter for each neighbor indicating the number of times it will allow the neighbor to use its link without the neighbor reciprocating. The counter is decreased when the neighbor first uses the link

during an advertisement period, and increased when the device uses one of the neighbor's advertisements. When the counter reaches zero, the link is no longer advertised.

The initial value of the counter must be positive and is a parameter of the algorithm. Larger values are more energy efficient as they reduce the need to switch between links; at the same time they can create short-term unfairness in link usage.

Traffic Optimization. Each device keeps a list of valid received advertisements that it can use to optimize different goals as follows:

- **Save Energy:** devices will always send traffic via advertised links; they will only use their own link when no other links are available. This is the default strategy.
- **Maximize Bandwidth:** devices will use all advertised links as well as their own in a bid to maximize the throughput they receive.
- **Minimize RTT:** the Save Energy strategy will also reduce the RTT because advertised links are by design in the high power state which gives small round-trip times. However, the "local" link used to reach the neighbor also adds precious milliseconds to the round-trip time; devices wanting the absolute smallest RTT might prefer their own link in the DCH state to advertised links.

When device A wishes to use neighbor B's link it will simply route packets via the "local" link to B. B will receive packets on the local link, change their source address and port (i.e. NAT) and place them onto its cellular link. Return traffic reaching B will go through the (reverse) NAT and then will be forwarded to A. In effect B is acting like a mobile hotspot for A - and this is already supported out of the box by most mobile platforms.

New TCP connections will work seamlessly over the new link, without the applications needing to know which link their traffic is taking. As long as these connections finish before the advertisement period ends, everything is great.

However, connections that last longer than the advertisement period will simply be terminated, as they are bound to the IP address of the advertised cellular link. This is a fundamental limitation of TCP: once a connection is created, it is bound to the addresses of the endpoints. Change the addresses and the connection will just die. As 97% mobile traffic runs over TCP [52] and many connections are long-lived, this limitation severely restricts the utility of the kibbutz.

Multipath TCP. To take full advantage of the kibbutz the newly standardized MPTCP protocol [50] is used. It is an evolution of TCP that allows unmodified applications to use multiple paths in the network through a single transport connection. MPTCP connections are composed of one or many subflows that appear like a single TCP socket to the applications. By default, MPTCP will use all available subflows to send data, favoring connections that have lower loss rates [153]. *With MPTCP, endpoints can add and remove subflows at any point during the lifetime of a connection*. Using this feature any connection can be redirected to use the

cellular or neighboring links by adding new subflows. To force the connection's packets via the new subflow there are two options: close the cellular subflow, forcing the remote end to use the new subflow, or announce the remote-end that the direct subflow has lower priority. The second option is used in the Mobile Kibbutz because it avoids repeated setup and tear-down of subflows over the device's cellular link. By default, all subflows over the device's link are marked as low priority when created. Subflows routed via neighbors are marked differently depending on the optimization goal.

In the energy saving profile, neighbor subflows are marked as normal priority, thus all traffic will prefer them to own 3G links; these will only be used when all the neighbor links experience repeated timeouts.

In the bandwidth and RTT profiles all subflows will be marked low priority, and transition between the two modes is done automatically: if the application generates enough traffic all subflows will be used thus maximizing bandwidth; otherwise, the MPTCP scheduler will send the packets via the available subflow with the smallest RTT.

Prioritization. All devices will prioritize their own traffic over that of the neighbors to ensure good user experience. Implementing prioritization for uplink traffic is straightforward by using priority queuing; on the downlink the kibbutz provider (see below) prioritizes direct traffic.

Bandwidth Watchdog. Consolidating many users' traffic onto a single cellular link may exceed the capacity of that link. The MPTCP priority mechanism will always prefer by design a congested kibbutz link to the direct link, but this may result in poor performance.

That is why we have optimized the MPTCP path selection algorithm for the kibbutz: if a high priority subflow delivers insufficient performance (determined by examining the *ssthresh* variable and the number of timeouts currently experienced), the mobile starts using its own low-priority cellular link - in effect it switches temporarily into bandwidth mode - until the kibbutz link recovers.

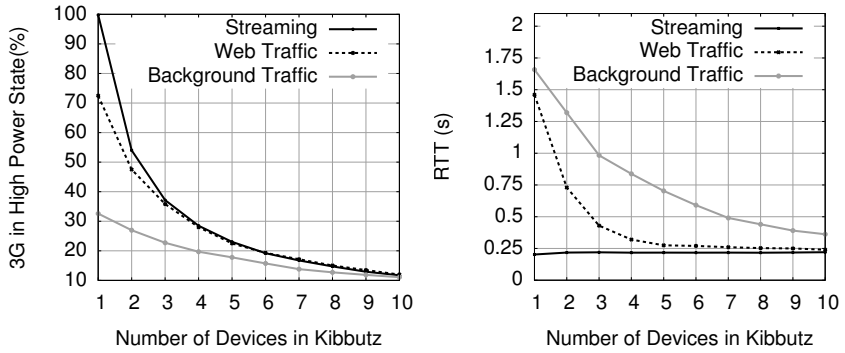
2.1.3.2.2.2 Deployment Considerations

To deploy the kibbutz, mobiles need to install the client app and an MPTCP patched kernel. Recent Linux kernels already support MPTCP that can also run on Android devices. The kibbutz implementation presented in next sub-section runs on Android and uses MPTCP capabilities.

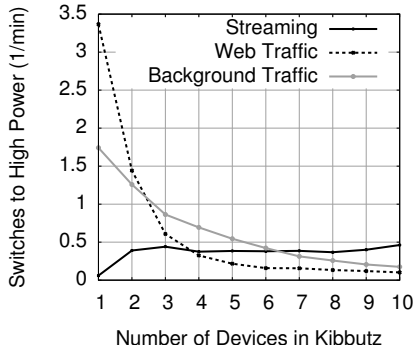
As MPTCP is not widely deployed yet, most servers do not support it, therefore a MPTCP proxy is needed to speak MPTCP to the mobile devices and regular TCP to the rest of the Internet, as suggested in [116]. The proxy is a standard Linux box running MPTCP. The service is supported by a kibbutz provider, where users will register using a unique identifier such as their phone, or SIM number. The kibbutz provider deploys a proxy in the wired domain, and the kibbutz client is pre-configured to route traffic via this server. As the kibbutz groups mistrusting users, the kibbutz provider handles security and bandwidth accounting.

2.1.3.2.2.3 Security and Bandwidth Accounting

To alleviate security and privacy concerns, such as traffic snooping or man in the middle attacks, Kibbutz members have to encrypt all traffic they forward via their neighbors. Equally important is that a users'



(a) Power Savings (b) Reduction in RTT



(c) Reduction in Signaling

Figure 2.14: Simulation results for the mobile kibbutz with varying number of devices and three classes of applications.

cellular data bill is not inflated by traffic forwarded for other in a Kibbutz. This can be implemented by tracking two numbers for each mobile using the Kibbutz service:

- *Bytes Given* B_G : measures how much a device has contributed to the Kibbutz by forwarding data for other users.

- *Bytes Taken* B_T: measures how much a device has used the Kibbutz by having other users forward its data.

As long as these two numbers are accurately measured, the Kibbutz provider can apply a number of billing policies including enforcing fairness ($B_G = B_T$), charging users for which $B_G < B_T$ and paying users when $B_G > B_T$.

A unified solution that addresses both security and accounting is proposed. The premise for this solution is that each mobile device shares a secret key with the Kibbutz provider; this is used to bootstrap authentication and security.

Consider the typical scenario where a mobile device A wishes to have a packet m forwarded via its neighbor B. Because A wants to stop B from reading its packet, it can just encrypt m with K_A and forward it to B, that will forward it to the provider. This simple solution ensures privacy, but does not offer enough support for accounting: the provider can see the packet is coming from A but it has no way of knowing that B forwarded m on A's behalf.

If the link between B and the provider is secure, it suffices if B adds his identity to the packet. If it is not, someone on the path can just substitute its own identity for B's, "stealing" the packet (from an accounting point of view). The solution, again, is for B to encrypt the whole packet with K_B. By decrypting the packet once, the provider knows B forwarded the packet. By decrypting it again, it knows A originated the packet. Can B charge A without A's knowledge? Freshness is a concern: B could replay the same packet over and over again, claiming it is performing work on behalf of A. To avoid freshness issues, all packets carry a 64-bit nonce; packets with duplicate nonces are dropped, and not accounted for by the provider. If A were malicious, it can exploit the nonce to make B forward packets that get dropped at the provider; this can be easily circumvented by letting B know when a number of packets have been dropped.
This solution has a few desirable properties:

- Traffic is protected against eavesdropping, impersonation and modification.
- User B cannot claim it has forwarded the traffic on behalf of user A unless A originates the packet, B receives it and forwards it to the provider. If many users overhear A's packet and forward it, the replay protection will ensure the packet is accounted for only once.
- User A cannot trick B into forwarding traffic for free.

Encrypted tunnels with replay protection are easy to setup, so are a good candidate to implement this protocols in practice. The proposed implementation uses OpenVPN UDP tunnels, but other tunnels such as IPSEC-ESP could be readily used. Each device will setup a tunnel to the provider to forward traffic on behalf of others, and a tunnel for each peer it intends to use [6]. The handshake needed to setup the tunnel is relatively cheap, and is amortized over the duration of the peering (several minutes in public transport [86]).

2.1.3.2.3 Simulation analysis

In this section the simulation is used to understand the basic properties of the kibbutz across parameters that cannot be easily controlled in practice.
A discrete-time simulator has been implemented for scheduled traffic requests from devices. To model 3G power consumption the simulator tracks whether a device is in the high power state; the tail timer is given as input. Cellular link speed is set to 2Mbps. RTT is measured for each traffic request; it is set to 2s if the device must transition to the high power state, otherwise it is set to 200ms. Finally, the simulator assumes the local links are perfect, having zero delay and infinite bandwidth. The simulator also implements the tit-for-tat algorithm and sets initial counter to five slots.
Three application classes have been modeled: streaming models audio and video streaming apps that send traffic every second thus keeping the cellular link up all the time; web browsing is modeled using randomized inter-arrival time with an average of 13s (value chosen by profiling our own browsing habits on mobiles); background traffic such as podcasts, or email and was similarly modeled with an average of 30s.

[6] In theory a single tunnel would suffice across all peers, but we have not implemented this optimization.

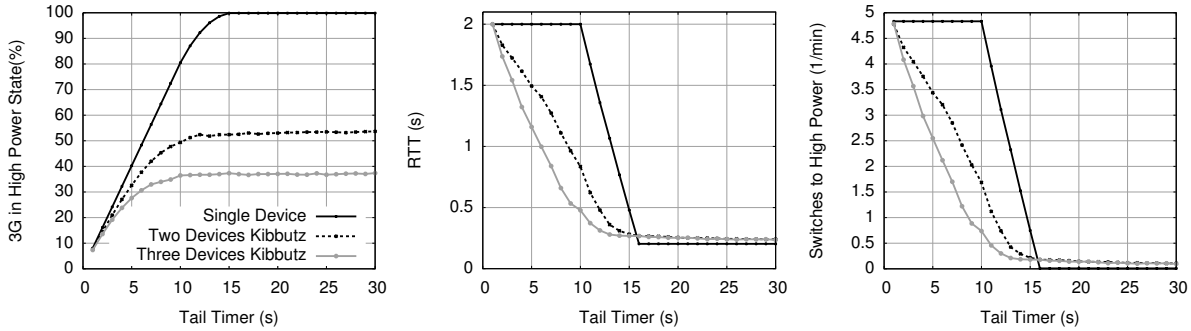


Figure 2.15: The Tail Timer trades device power for reduced signaling and RTT (web-like traffic).

Energy Savings. To understand the possible benefits of the kibbutz, the number of devices have been varied and the three app types described above have been ran. In Figure 2.14(a) the expected energy savings from a kibbutz are shown; the numbers given represent the percentage of time the 3G interface stays in the high power state. In the best case, the power can decrease inversely proportional to the number of devices.

Streaming keeps the radio up all the time if a single device is playing; if two devices are playing in a kibbutz, they will each save around 45% of the 3G energy. As more devices join the kibbutz, the benefits increase as expected. A similar curve is seen for web traffic; here the baseline is at around 80%, because the radio manages to sleep a little in between two pages. For these two applications even small numbers of devices make a big difference to power consumption. The story is different for background traffic; here the baseline consumption is only 30%, and it drops more slowly; with three devices it stays at 24%.

How many devices is it feasible to have in a kibbutz? Assuming the device density is high enough, the local links will become bottlenecks when many devices join. Experiments with two Android devices have been performed without any problems. Going to four-five devices is certainly feasible for Bluetooth links as long as aggregate traffic rate is low. Beyond that other costs may start dominating and we will see diminishing returns. With this in mind, it becomes apparent that the kibbutz does not bring many benefits for background-style traffic: the messages are so sparse in time that consolidation does not help much, especially for small numbers of devices. It makes sense, then, to only enable the kibbutz when the device is active.

RTT Reduction. 3G state transitions induce high delays to packets; the measurements show this delay to be around 2s depending on time of day, for a number of operators. Once the radio is in the high power state, delays are much lower - typically 100ms-200ms. The average RTT experienced by each transfer for each app class has been measured, with the results shown in Figure 2.14(b). The simulated RTT decreases five times for web transfers with three devices in a kibbutz. As web transfer is RTT-bound, this reduction is significant and will enhance user experience. Background traffic is less sensitive to RTT, hence the decrease in RTT while sizeable it may not matter that much in practice. Finally, RTT increases a little for streaming apps because the kibbutz switches between different links that will have to transition to the high power state. With a single device this is not needed, as the link is up all the time.

Reduced Signaling Costs. The previous measurements focus on user-centric performance indicators. The

aim is also to understand how the kibbutz affects the network, in particular in reducing the signaling costs between the devices and the cellular base-station. This signaling is an important optimization target for carriers [112]. To this end, in Figure 2.14(c), the frequency of transitions to the high power state induced by our three categories of apps has been measured. Web and background traffic induce a transition every time a packet is sent when a device is running standalone. In a three device kibbutz the number of transitions is reduced tenfold for web-traffic, and halved for background traffic. Streaming will incur a bit more state transitions with the kibbutz, because instead of staying always up, the 3G link is now down when the mobile consolidates its traffic with the neighbors.

Effect of the Tail Timer. The idea is to understand what effect the tail timer has on the key metrics for users (power) and the network (signaling). The tail timer is the only knob available to mobile operators today. To measure its impact, the tail timer is varied from 1 to 30s while measuring the web application when running standalone and in groups of two or three devices.

The tail timer offers a trade-off between power on one hand (Figure 2.15(a)) and signaling (Figure 2.15(c)) and RTT (Figure 2.15(b)) on the other. The graphs show how difficult it is to choose a good trade-off today: one has to go for low signaling and delay or low power, but not both. The kibbutz changes the trade-off space: it tames power consumption for higher values of the tail timer, while also reducing RTT.

If we compare the tail timer to the average inter-arrival time between web requests (13s) we get other insights from these graphs. When the tail timer is very low, web traffic rarely benefits from the kibbutz because the radio is in the low power state most of the time. As the tail timer increases, the cellular link is up more and this allows the kibbutz to consolidate traffic and obtain benefits for both power and RTT. As the tail timer goes beyond 13s, the kibbutz only improves power usage.

Dynamic Adjustment of the Tail Timer. There is a wealth of existing research that exploits fast dormancy, a feature standardized by 3GPP [1] that allows the mobile device to demand a transition to the IDLE state, thus reducing its tail time and its associated energy cost. Using fast-dormancy can save energy but it requires that the device knows the traffic pattern of the application [112, 111, 33]. A concrete example is the MakeIdle algorithm proposed in [33] that uses machine learning to infer when it is appropriate to transition to sleep.

To understand how the Kibbutz compares to these works, an oracle that knows the traffic workload and decides after each packet sent whether to transition to sleep or not has been implemented. The oracle goes to sleep only if the next packet is due after a threshold number of seconds. The threshold takes into account the time needed for the two state transitions, each of which takes 1-2s in the measurements; it is set to 4 or more seconds.

The simulations reported in Figure 2.14 have been ran with the oracle algorithm and varying thresholds. The algorithm has no impact on the streaming app, and the performance is the same as on a standard mobile device. This is because streaming packets have low inter-arrival times and there is no time to sleep. In comparison, a three-device kibbutz cuts energy consumption to a third.

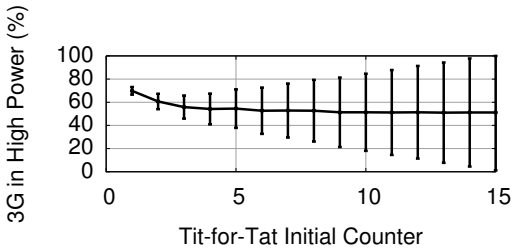


Figure 2.16: Power consumption distribution for a 5 minute streaming session.

The web traffic results are more interesting. With a 4s threshold, the oracle reduces the time spent in the high energy state from 72% to 17%, but the energy reduction comes at a cost: the average RTT increases from 1.46s to 1.9s as the number of state transitions increases by a third. This should be avoided as it affects the scalability of 3G networks [112, 2]. In fact, carriers are afraid of enabling fast-dormancy because this could overload their networks [2].

To reduce the effects of signalling, the oracle algorithm's switching threshold is increased to be the same as the tail timer (10s): this guarantees that the number of state transitions (and the RTT) will not increase. In this case, the RTT and signaling costs stay the same, while the time spent in DCH drops from 72% to 36%. In comparison, a three-device kibbutz has similar energy behavior (35% in DCH) but it reduces the RTT to 0.4s and the signaling costs by 50%.

Tail reduction has the biggest positive effect on background apps, decreasing the time spent in high power state from 33% to around 8%. For such apps the Kibbutz gains are modest: a three-mobile kibbutz reducing energy to 22%.

The implemented oracle algorithm provides an upper bound for tail-reduction algorithms, and the actual benefits will be smaller in practice. Nevertheless, this comparative analysis shows that the Kibbutz excels for streaming like applications, while tail-reduction works really well for background applications—hence the two techniques could complement each other nicely. For web-style traffic, the energy gains are similar, but the Kibbutz also reduces the RTT and signaling costs.

Fairness. All the experiments shown so far were run for a long period (2 hours); this ensured that all the performance measurements were nearly identical across all devices.

The tit-for-tat counter governs how many times a device will share its connection without the neighbor sharing back. To understand the effect of this parameter the streaming application has been ran for a short period of time (5 mins) with different values of the counter. The results in Figure 2.16 show that using small values of the counter leads to increased energy consumption because of frequent switching between power states and the tail timer. Higher values of the counter are more efficient, but lead to unfairness: a single device may bear the whole cost of cellular connectivity for high values. Values above five give diminishing returns in average consumption, but increase unfairness. This is the value used in all other experiments.

Interactions Between Different Application Classes. Table 2.2 shows the experimental results involving different classes of applications. A table row shows an application class running in a kibbutz of two devices

App. Class	Back.	Web	Stream.	All
Background (30%)	25%	21%	21%	17%
Web (80%)	68%	49%	48%	41%
Streaming (100%)	78%	58%	54%	50%

Table 2.2: Kibbutz experimental results with different classes of applications.

with another application class (the column), or in a kibbutz of three devices. Each cell lists the amount of time the 3G interface stays up; the baseline without kibbutz is also given on each row. For instance, the Background power consumption drops from 30% to 21% when running in a kibbutz with a Web application. The numbers show that the application mix does influence the results but not dramatically. Also, the savings are shared between the different apps. As expected, the Background class of applications improves the least.

2.1.3.2.4 Implementation

The Kibbutz has been implemented using Bluetooth and WiFi as local links. First the devices have been enabled to seamlessly route traffic for neighbors with pre-existing Android mechanisms: BNEP interfaces have been created using `pand(1)` from the BlueZ[7] suite available on Android 4.1.2 by employing normal netfilter mechanisms to route traffic and perform Network Address Translation.

The traffic consolidation scheme and the short-term fairness are implemented in a Linux kernel module. Netfilter hooks are used to efficiently handle the traffic on every interface without polling. This way, the module knows when the mobile link is up, when a neighbor's link is used or when a neighbor is using our own link.

The main part of the service consists in "moving" traffic from one link to another, without changing the application. This is achieved using standard MPTCP mechanisms. Upon system startup the cellular link is set to low priority and MPTCP is disabled for local links. This ensures MPTCP will try to avoid the 3G link when emitting traffic and will never attempt to create flows on Bluetooth links by itself – by default MPTCP creates flows on all interfaces. The module exposes an interface permitting userspace program to declare the local links as "usable". Once a local link (Bluetooth or WiFi) is declared usable MPTCP will consider it. Since the cellular link has low priority, any local links will win the flow selection process, i.e. traffic will move from the mobile link to the local links. Implementing the bandwidth maximizing / RTT minimizing profile is easily achieved by removing priorities and using the default scheduling algorithm of MPTCP. For the energy saving profile, a bandwidth watchdog has been implemented by changing the MPTCP scheduling algorithm.

The upper part of the kibbutz has been implemented as a userspace driver that uses the kernel module and controls the entire process. This driver monitors the local link and the kernel module for advertisements. The kernel module sends up notifications like 'when did neighbor A last use our link', 'when did we last use A's

[7] http://www.bluez.org/

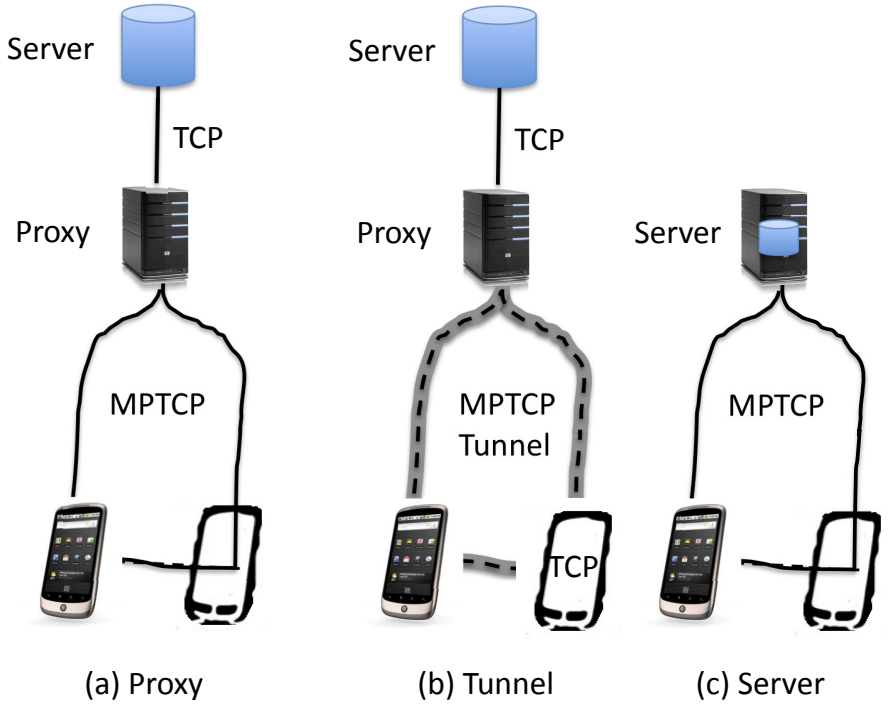


Figure 2.17: Experimental setup

link', allowing the enforcement of short-term fairness via a tit-for-tat mechanism.

2.1.3.2.5 Experimental evaluation

The mobile kibbutz has been deployed on two Galaxy Nexus mobile devices running Android 4.1.2 (Jelly Bean) and tested with synthetic traffic as well as representative applications. Power usage has been measured using the Monsoon monitor. Most of experiments have been ran using the only cellular operator avaliable that allowed MPTCP (it did not remove MPTCP options from packets). This operator offered 3G (HSDPA) at advertised speeds up to 7.2Mbps.

To enable the kibbutz, a Linux box running MPTCP, that is used in all experiments, has been also deployed. The UDP tunnels are used for security in all setups, but are not shown for simplicity.

The setup is shown in Figure 2.17. Ideally, this box would act like a SOCKS proxy terminating MPTCP connections and opening TCP connections to the real server (Figure 2.17.a). For this to work a proxy support would be needed from Android; unfortunately this is only available over WiFi connections, not 3G ones – this seems to be a policy option rather than a technical one. A work-around is to open an MPTCP-based tunnel to the proxy, and tunnel all device traffic (Figure 2.17.b). In this case, the MPTCP-based tunnel is also encrypted, replacing the UDP tunnel. This setup works fine, however tunneling TCP connections over (MP)TCP can lead to bad performance interactions that could bias our experiments. That is why the third setup is used (Figure 2.17.c) for most of the experiments: here the box also serves content, running stock HTTP and streaming servers.

In most of the experiments Bluetooth is used as the local link, as it has low, symmetric energy consumption across connected devices. The kibbutz has been also successfully tested with WiFi connectivity, with one device acting as an access point and the other as client. This setup offers better performance at the cost of

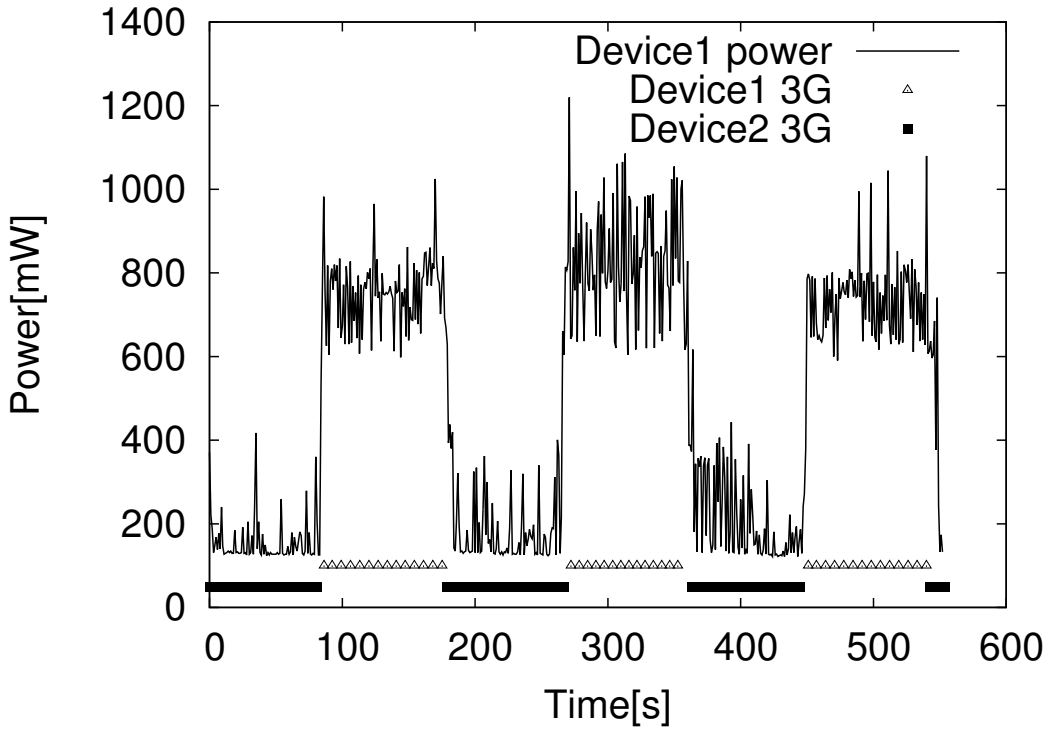


Figure 2.18: Simple TCP ping with a period of 0.5s on two devices in a kibbutz.

increased energy consumption; it was used for the download tests.

The next subsections describe the practical findings. In brief, it confirms the simulation results showing that the kibbutz significantly reduces the power costs associated to using the cellular links and their RTTs. In addition, network-related costs make up a large fraction of device consumption: using the mobile kibbutz reduces total phone energy consumption by 25% for radio and web browsing.

2.1.3.2.5.1 Basic Functionality

To test the basic operation of the kibbutz, some experiments have been ran with each device acting a simple client that opens a long-running TCP connection and then sends a timestamp twice per second to the server which just echoes it back. The clients measure the app-level RTT and print it.

In Figure 2.18, the upper curve shows the power consumed by Device1 as it switches its 3G card on and off. The segments at the bottom correspond to the packets received at the server from Device1 or Device2. The upper segment sequence corresponds exactly with the power consumption recorded by the power-meter, confirming the fair behavior of the kibbutz. If a device uses only its link, it needs 754mW on average, whereas it only needs 169mW if it uses its neighbor - the upper and lower levels of the power measurement. The average power consumption of Device1 is 480mW. This is a saving of 37% over its maximum consumption if it used only its 3G connection.

2.1.3.2.5.2 Improving Popular Applications

Four applications that depend on network connectivity have been tested: audio streaming (e.g. radio, VOIP), video streaming, web browsing and app downloads. While not exhaustive, we believe these capture the main "active" traffic patterns on mobile devices. The focus is on active apps, leaving out background apps such

as Facebook or email that generate traffic rarely. A worthwhile improvement is not expected for such apps, especially when consider the overheads for running the kibbutz is considered.

Streaming Apps. Apache was installed on the server and VLC player on the mobile phones. For audio streaming MP3 files encoded at 48Kbps was used. The Galaxy Nexus phones consumed 978mW when streaming radio alone, with the screen off. Most of this cost is due to the 3G link. When running a two-device kibbutz with both streaming the same file, instantaneous power consumption was bimodal: the device routing data consumed 1047mW (slightly more than standalone), while the other device consumed only 403mW. The two devices traded roles periodically.

The average power consumption for a device in the kibbutz was 723mW, 25% less than standalone consumption. Put another way, a device streaming radio in a kibbutz will last 34% longer than one running standalone. The savings in absolute terms—around 250mW—are a bit less than half the energy consumed by the 3G radio in the high power state (650mW) which is the theoretically maximum gain that can be had in this configuration; this confirms the kibbutz is working as expected.

Video streaming was tested with a Youtube clip encoded at standard definition (250Kbps). Here standalone energy consumption was 1819mW - this value includes the cost of the screen, as well as of the GPU used for rendering. Consumption was again bimodal in the kibbutz: 1890mW for the forwarding device, and 1205mW for the other. Average consumption for devices in the kibbutz was 1547mW—15% less than standalone. A device streaming video in a kibbutz will last 18% longer than one streaming on its own.

Web browsing. Popular web pages today are optimized to the point where their load times are determined by the connection RTT rather than the bandwidth. That is why Google has been actively promoting a number of TCP optimizations aimed at reducing the number of RTTs required to download a webpage, including a proposal to increase the initial TCP congestion window to 10 packets [29] and to implement a flavor of transactional TCP called TCP Fast Open [28]. On cellular links, the RTT after idle is very large, dominating the load time. Can the kibbutz help alleviate this problem?

Measuring web traffic is tricky. Webpages cannot be simply replicated on the server because modern sites link many objects, many of which are dynamically generated and on different sites. That is why two types of web browsing experiments have been ran: an MPTCP-based tunnel and emulating web-traffic. The tunneled experiment has a few potential pitfalls: it is prone to TCP-over-TCP performance issues, and it uses a "visual" method of measuring the load time. The emulated web traffic experiments avoid both problems: they give precise timings and avoid encapsulation problems, but may not model web traffic as accurately. Both experiments showed good improvements for the kibbutz, which gives confidence that the benefits will appear in practice.

In the tunneled setup (shown in Figure 2.17.(b) d a Google search page was repeatedly loaded using a single device over 3G, or a device in a Kibbutz using its always-available neighbor's advertisements. In both cases the screen is on and the device's browser renders the page. To measure delay, the power meter readings was

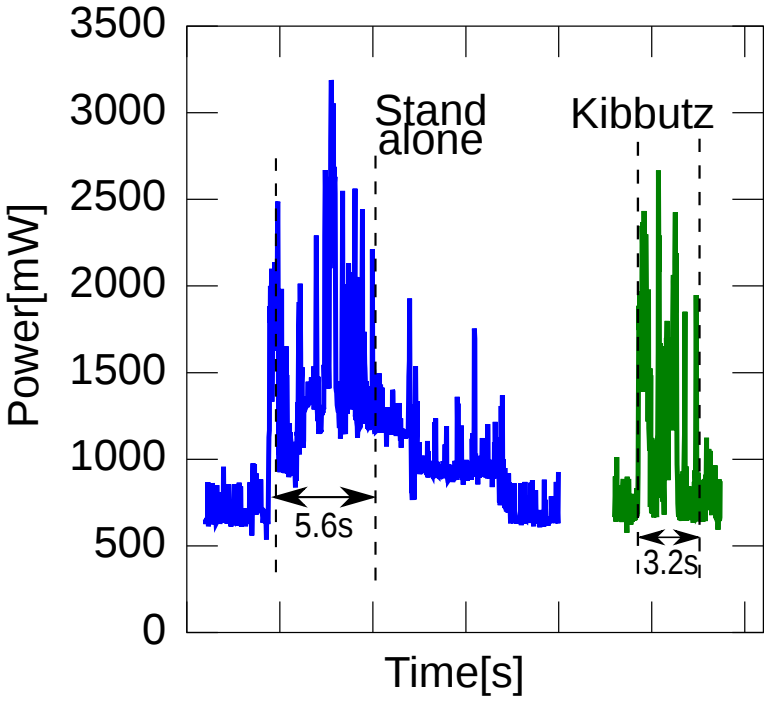


Figure 2.19: Power measurements while loading a Google search results page directly or with a kibbutz.

used, identifying the "mountains" of consumption that appear during page load. These are shown in Figure 2.19. Downloading and displaying the Google search page takes 5.6 seconds on average, and it costs around 15.9 Joules. When using a kibbutz, browsing the same page takes 3.2 seconds and costs 3.9 Joules. The kibbutz loads the search page 2.4s quicker, just a bit more than the state promotion time measured for this provider.

The emulation of web traffic was done with *tcpdump* traces obtained by loading typical pages from several mobile optimized web sites including BBC, CNN, Amazon, and Google search. Most sites exhibit the following behavior: after a DNS lookup, a main index file of 20K-50K is started; smaller transfers of 1K-5K of images, ads, or scripts get downloaded progressively as the main index is parsed; about halfway through, another DNS lookup is made, usually to a CDN server, and more small transfers occur. Most newspapers and google searches include another larger transfer of 20K-50K, with some images. For all the sites used, the browser and the server use HTTP 1.1 with persistent connections, therefore one transfer typically brings in several smaller files. A random mix of transfers was also generated by mirroring each of the observed patterns interspersed by a random 10-15 seconds period of silence.

Figure 2.20(a) plots the cumulative distribution for the completion times of 50 web-like downloads as described above. Compared to a standalone user, a user in a kibbutz receives a page 1.2s sooner on average, and almost 4s sooner for the 90%. The bulk of this saving comes from avoiding the ramp up time when a neighbor already has the 3G active. Web browsing via the kibbutz also saves power: while a standalone user needs on average 435mW to download the page, the user of a 2-kibbutz only uses 297mW, 31% less.

App Downloads. How long does it take to download an average mobile application alone or in a kibbutz? The same 6MB file was repeatedly downloaded from the server (this is the average size of Android apps

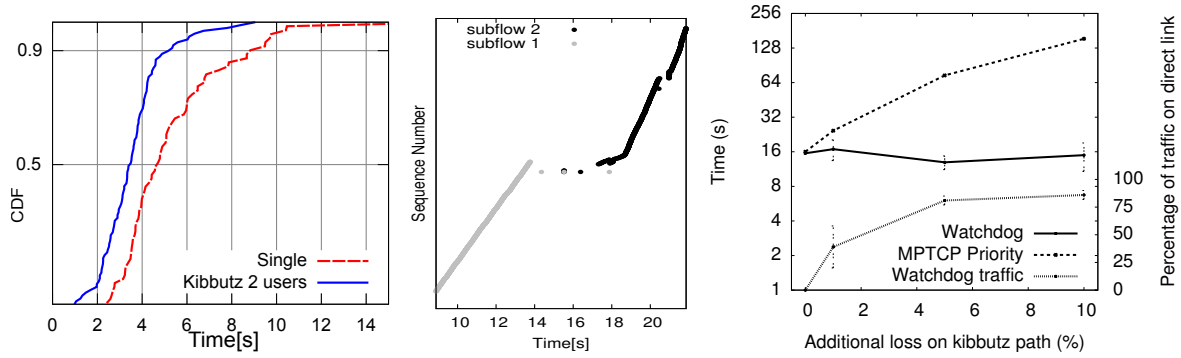


Figure 2.20: a) Time to load a mobile optimized web page. kibbutz needs 1220ms less at the 50% and 3980ms at 90%; b)The Kibbutz reacts when a neighbor stops forwarding; c) The watchdog ensures good performance when the kibbutz path is congested.

[3]), measuring download time and energy consumption. For a standalone device, the download time was 31 ± 8 seconds. The high variance is due to competing traffic in the HSDPA network used. The phone draws 1.3Watts during this download, consuming on average a total of 41 Joules.

For the kibbutz, one device generates low-rate traffic using the synthetic client. The other device is running in bandwidth optimization mode, using all links it has available. This kibbutz uses WiFi as the local link to ensure the best throughput (the bandwidth-hungry device acted as client, and the other acted as Access Point). HSDPA dynamically assigns all available cell bandwidth to active users, so joining two users connected to the same cellular base station in a kibbutz will not always increase the total bandwidth available to them, yielding similar download times to the standalone test. If the two devices are in different cells or sectors, bandwidth should always increase.

This hypothesis in confirmed by using a different operator for the second device - this operator strips MPTCP options, so MPTCP traffic was tunneled over UDP. Downloading the same file in the kibbutz takes 21 ± 5 seconds, decreasing download time by a third on average. Counter-intuitively, the kibbutz also saves power despite using more network interfaces: the total energy consumed to download the file is 31 Joules, 25% less than standalone.

2.1.3.2.5.3 Robustness

If a mobile device leaves the area of a Kibbutz, how will it affect the performance of other Kibbutz users? To answer this question an experiment where one user is downloading a large file via its neighbor was also ran. While the transfer is taking place, the neighbor suddenly stops forwarding packets. Figure 2.20(b) shows the progression of the connection sequence number against time, with the disruption happening around second 14.

The watchdog mechanism quickly detects after one timeout that the prioritized path has poor performance, so it starts sending packets via the direct path. After another few seconds—needed for the idle 3G link to become active— traffic resumes via the direct path.

Sudden path blackouts as above will be relatively rare: it is more likely that a path will become worse as the user is moving away from the kibbutz, leading to increased packet loss rates. High packet loss rates may

appear also when there are too many users are sharing one link at the same time.

To understand how the kibbutz fares in such cases, one device downloaded a 2MB file via its neighbor which also dropped a fraction of the packets it forwarded. Figure 2.20(c) shows how the total download time varies as the loss rate varies. Vanilla MPTCP always prefers the path via the neighbor (top curve), which dramatically increases download times when loss rates are high. The watchdog detects performance problems and starts using the direct path, managing to have similar performance regardless of the loss rate. Moreover, the fraction of bytes sent via the direct path: when there is no loss all bytes are transported via the neighbor. When the loss rate increases, most bytes are sent using the direct link.

These experiments outline the importance of using MPTCP for load balancing across links: single path mobility solutions must commit to a single path, while MPTCP can dynamically choose the best path at any time.

2.1.3.2.5.4 Kibbutz Overheads

The security mechanisms have relatively low overhead, as symmetric key operations are cheap and the data rates are comparatively low. The experimental results were measured with traffic flowing over encrypted tunnels, yet the energy, rtt and bandwidth improvements were strong.

The kibbutz brings benefits when there is cellular traffic, but how much does it cost when the phone is idle? A furhter experiment has been carried out to measure how much an idle device running the kibbutz software consumes compared to an idle device without the kibbutz. Higher costs are expected due to Bluetooth networking and receiving kibbutz advertisements.

Galaxy Nexus phone draws 12mW with screen and all wireless links except 3G off (3G was in the IDLE state). The same phone running the kibbutz draws 60mW - the kibbutz adds 48mW to the baseline power consumption. The power usage was broken down as follows: starting Bluetooth raises consumption to 28mW, and enabling IP connectivity over Bluetooth (Pan) further raises consumption to 50mW. Finally, receiving kibbutz advertisements increases to 60mW.

The kibbutz brings most benefits for applications that send traffic at short intervals. Background traffic benefits less from the kibbutz: a simple optimization is to run the kibbutz when the traffic rate is greater than some minimal threshold. This reduces baseline costs to almost zero, while giving the same benefits for busy applications.

2.1.3.2.5.5 Kibbutz with LTE

The proposed kibbutz solution has not been ran over LTE due to a lack of LTE deployments. However, the results obtained from simulation and practice, along with published measurements of LTE networks, can be used to predict the benefits that can be had in such scenarios.

The 3G experiments showed that streaming apps saved approximately 40% of the 3G power. If the same apps are ran on LTE similar percentile savings are expected: total savings should be bigger because LTE interfaces draw between 1 and 3 watts [69]. LTE's state promotion is quicker than 3G's being between 500ms and 1s

[33], and the RTTs are around 50ms. We thus expect the delay gains from the kibbutz to be a bit smaller. Finally, the bandwidth gains can be much bigger from LTE, as long as the kibbutz users connect to different cells and WiFi is used for local connectivity.

The importance of cellular links to the mobile experience has long been recognized, and the result is a wealth of research measuring the properties of cellular connectivity [112, 69] and proposing solutions to optimize them (e.g. [111, 12, 125, 33] to name only a few). Most of this research has focused on reducing energy consumption of cellular links by adopting one of two high-level approaches.

We have already discussed in our simulation analysis techniques to reduce the tail timer via fast-dormancy, e.g. [33]. The second approach is to use application knowledge to optimize the 3G link; for instance, [156] instruments email clients to batch operations to network and flash memory. Using application knowledge, transmissions from different applications are batched by delaying subsets of traffic to reduce the tail energy [12] or to use the cellular link when it is cheapest [125]. Batching only caters for background, delay-insensitive traffic: either apps must be changed to take advantage of it, or the system must infer which traffic can be delayed.

The mobile kibbutz allows energy optimization across many devices, and is complementary to existing works. It offers the biggest benefits for applications that send data (almost) continuously such as audio and video streaming, as well as web browsing. These applications are highly popular on mobiles, yet they are not supported by existing works. Beyond energy, the kibbutz significantly lowers the impact of state promotion delays and reduces signaling overhead - to our best knowledge, ours is the first work concurrently reducing power, RTT and signaling cost for cellular links.

The basic idea of the kibbutz is to opportunistically use other devices' links, and this idea is not new: Shair proposes to use other mobile users' free minutes and texts via local connectivity [70]. Recently, it has been proposed to utilize idle 3G links to overcome access links bandwidth bottlenecks [140]. The kibbutz is similar in spirit with British Telecom's FON network [91] where users share their DSL links with each other via WiFi. Sharing in the kibbutz is much more dynamic and the gains go beyond the basic connectivity, reducing both power consumption and delay.

2.2 Polyversal TCP

Polyversal TCP (PVTCP) is a new mechanism to provide bandwidth liquidity by pooling different data transports including sockets, shared memory or any other communication means. This research is headed to investigate how to improve transport protocols in the Internet and to provide the universal transport framework to fulfill the requirements of modern applications and fit the architecture of the modern data centers. Research efforts have been carried out recently to improve the architecture of the transport layer. However, this work introduces the *polyversality* definition to describe the property of transport stack to work independently from individual protocols hiding all low level details from applications. This, for example, includes network and high performance local communications, such as shared memory channels or pipes, that work similarly to

the network connections providing the universal data transfer environment. There are a number of challenges to overcome including addressing and naming issues[139], middlebox influence[67][26], congestion control fairness[16] and compatibility issues.

2.2.1 Related work

Several studies which investigate ways to implement different sorts of polyversality in the transport layer have been published recently.

2.2.1.1 Name based sockets

Name based sockets[139] is a proposal to hide address families from applications. This work introduces an extension over the sockets layer which allows to use names instead of addresses. Moreover, it enables the support of multihoming and address migration in IPv6 networks by using IPv6 extensions (which may be filtered by middleboxes though). However, this research has nothing related to unifying transport families and is still mainly oriented on TCP sockets.

2.2.1.2 Protocol independent transport API

Protocol independent transport API[148] is the recently carried project of unifying transports API used for the data transmission. It proposes a new socket API which is not fully compatible with BSD sockets but is more clear in terms of transport selection. Moreover, protocol independent API suggests to use parallel probing for individual transports, name based sockets and the QoS transport API proposal. However, this work is concentrated on network data transfers and does not take local and non-IP communications into consideration.

2.2.1.3 ZeroMQ

ZeroMQ[71] is a transport framework that provides assured messages delivery over TCP sockets, Unix sockets and between threads of a process. It provides zero-copy transport between threads running in a single address space but not between processes however. Nevertheless, ZeroMQ does not hide transport interaction from applications and still requires modifications in applications to migrate from sockets to ZeroMQ API.

2.2.1.4 Megapipe

Megapipe[59] project tries to reduce socket overhead, especially for SMP systems. It presents lightweight sockets that are bound to a CPU and perform all data transfers via these objects. The main goal of Megapipe is to decouple sockets from the virtual filesystem layer. However, transport selection, naming and copying issues are not resolved by this project.

2.2.1.5 Conclusion

The analysis of the related work shows that this topic is widely investigated. A great number of attempts have been made to unify transports and addresses. Protocol independent transport API is the closest work in this area which describes and analyses a lot of properties of the universal transport framework. The aim of this research is to extend these ideas to create an improved transport architecture that can take the advantages of

ipc data transfers, use DNS for reducing protocol probes count and for better addresses migration support. Moreover, the polyversal transport framework should be able to adopt to the current system's architecture (for example, to NUMA topology) to provide maximum throughput and minimum latency for applications. The other goal is to keep the maximum possible level of compatibility with BSD sockets API to avoid the problems with application integration.

2.2.2 Progress to date

The first part of the study was mainly concentrated on two research projects:

- *ltproto* library - a prototype for studying the ipc performance in Unix;
- in-kernel prototypes of the polyversal transport stack.

2.2.3 Ltproto library

2.2.3.1 Overview

The initial efforts were headed to implement a transport that can incorporate various mechanisms that are capable of transferring data between applications on the same host. This work was heavily inspired by the previous research project called Fable IO. The goal of this project was to examine the existing ipc methods in Unix and to determine their adequacy for data transfer purposes. This work brought the *ltproto* library as the initial prototype of the polyversal transport framework (pvtf) itself. *Ltproto*, being a shared library, operates completely in the userspace and provides an intermediate layer between applications and IPC interfaces, such as sockets, shared memory segments and pipes. *Ltproto* can select the appropriate transport transparently and fallback to TCP socket if needed. Actually, the backup TCP connection is always kept open to provide a reasonable fallback if the other transports fail to operate, for example, in case of service migration in a cloud, when a service is moved from a local to a remote host. *Ltproto* library has a modular structure and allows the addition of new transports easily. Moreover, it provides a BSD sockets compatible interface (that currently supports only a blocking interface) which makes it possible to integrate the library with network applications that use BSD sockets without any changes by using LD_PRELOAD technique, for example.

2.2.3.2 Research goals

The main purpose of writing *ltproto* library was to set a base for further work on the pvtf, and to estimate and analyze the performance of Unix ipc methods in comparison with the traditional sockets and pipes in terms of throughput for different payload types and data transfer latency. These two properties are very important for high-performance applications and thus should be considered when selecting an appropriate transport.

The other goal of this research was to compare different operating systems in terms of their ipc performance. As a consequence *ltproto* was designed to be a cross platform library. At the moment, *ltproto* lacks functionality to be treated as the full prototype of pvtf. For example, it cannot perform name resolving and it doesn't support non-blocking IO which is widely used in high-performance environments. Nevertheless, *ltproto* demonstrated that traditional sockets and pipes perform worse than shared memory rings in most of

cases. Furthermore, it provided evidence of various circumstances that affect data transfer properties.

2.2.3.3 Results analysis

During the experiments with the ltproto library, we have obtained results from many systems, including both NUMA machines with 48 cores and traditional UMA ones with 4-8 cores. We have measured both throughput and latency for different ipc data transfer methods either standard, such as sockets and pipes, or shared memory rings based based on memory mutexes (futexes in Linux and umtxes in FreeBSD).

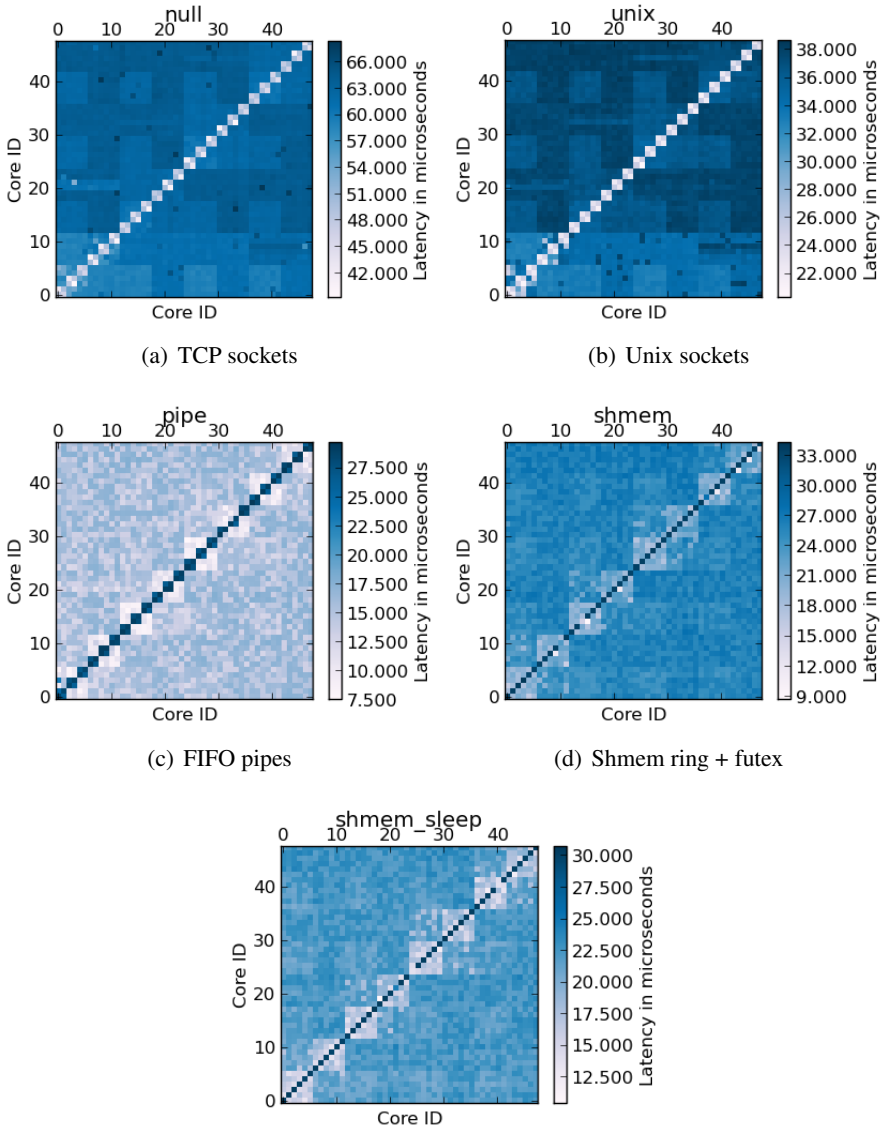
Moreover, the influence of the specific topology has been studied to demonstrate how CPU binding affects the data transfer parameters. This influence is especially significant for NUMA nodes, where each CPU cluster has its own memory controller, and transferring data between NUMA nodes is noticeably slower than inter-node communication. The same argument is valid for latency which is higher for data transfers between nodes. Figure 2.21 demonstrates the latency measurements for NUMA host with 48 AMD CPU cores on board.

This figure shows that there is a significant difference in latency depending on NUMA topology for different ipc methods. However, for shared memory rings the NUMA topology seems to have no influence, as it is likely impossible to bind shared memory to NUMA nodes properly. This behaviour is a target for further investigations.

Furthermore, the impact of system architecture is different for virtualized environments where a hypervisor has its own algorithms for memory and CPU time distribution. The results of throughput measurements are displayed on the Figure 2.22. These tests were performed on 2 machines running Linux kernel 3.5. The first one is an UMA Intel Xeon machine running inside the Xen hypervisor and the second one is an AMD NUMA machine with 48 cores. The performance of the NUMA machine is surprisingly low and it is required to perform explicit binding to NUMA nodes in order to to improve the performance. This experiment demonstrates the problems in dealing with the NUMA systems by an OS tasks scheduler. The last test was executed on the same hardware platform as the first one (UMA Intel) but with a different OS running - FreeBSD 10-CURRENT. This figure demonstrates the difference between ipc performance in these two OSes. Specifically, UNIX and TCP sockets in FreeBSD perform worse, whilst the scheduler seems to be slightly more efficient than in Linux, which causes the sleeping shared memory ring to perform better.

As a consequence, this research is also headed to investigate how we can improve the performance of ipc for NUMA enabled systems. And it is fair to say that the task of optimizing data streams for the specific topology should definitely be performed by the transport layer, as it has the whole understanding of the architecture of a system and can adopt data flows dynamically to optimize data transfer properties.

The other experiment that has been carried over the first year is the investigation of shared memory rings. This experiment was headed to study how could we optimize the data transfer over shared memory rings. In particular, each ring has a number of data slots and each slot may contain some amount of data. For a NUMA machine, there is practically no influence of these attributes on the final throughput, as shared



(a) TCP sockets

(b) Unix sockets

(c) FIFO pipes

(d) Shmem ring + futex

(e) Shmem ring + sleep

Figure 2.21: Latency test results for ltproto

memory is likely distributed over multiple NUMA nodes and the access time to a foreign node's memory has significantly more influence on the throughput than ring or node sizes.

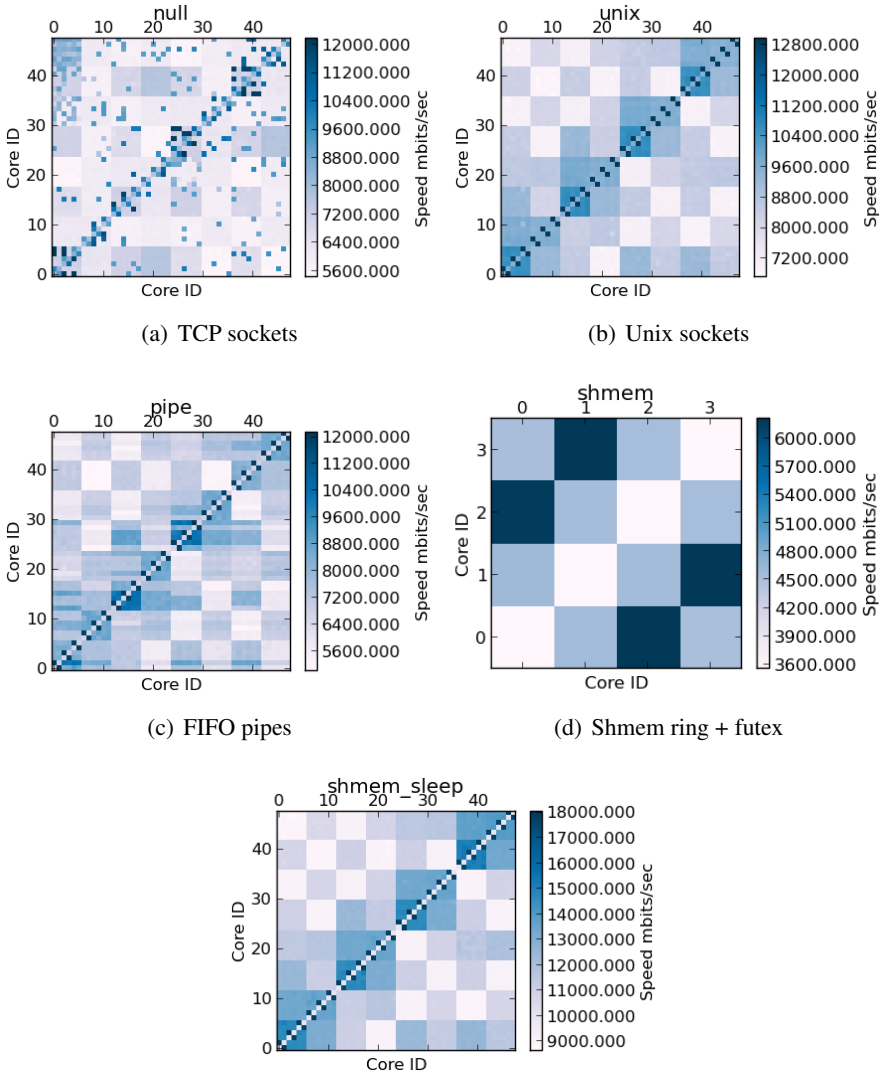
All these observations demonstrated that it is required to optimize data the transfer performed by Unix ipc, and now we have a set of attributes to be investigated, such as binding to specific CPU cores and NUMA nodes, selecting the optimal ipc method and optimizing this method for the specific system architecture.

2.2.4 In-kernel prototypes

Along with the experiments with ltproto library which operates completely in the userspace, several kernel level based prototypes have been analyzed.

2.2.4.1 Using netgraph for transports

Initially, we have tried to implement the transport framework based on the netgraph[39] system in the FreeBSD kernel. However, this prototype was not successful as netgraph lacks flow control and state man-



(a) TCP sockets

(b) Unix sockets

(c) FIFO pipes

(d) Shmem ring + futex

(e) Shmem ring + sleep

Figure 2.22: Latency test results for ltproto

agement. Flow control is essential for transport protocols as it prevents packets from being silently lost on a path, among other things. Moreover, netgraph has no methods to pass state data between graph nodes making it thus impossible to handle the states of different connections efficiently. Nevertheless, netgraph (as well as the Click router project [78]) has a modular architecture which could server as a base for future transport frameworks, as it allows to share common functionality between transport protocols and data transfer means. For instance, it is possible to have a plugin for joint congestion control or for distributing traffic over flows in a multipath connection. The overall architecture of the proposed solution of polyversal transport framework is based on a netgraph like interface.

2.2.4.2 Multipath transports inverstigation

The other experiment that was carried over the first year of study was the attempt to study and modify multipath transports, such as multipath TCP and SCTP (and address autoconfiguration specifically) to communicate with the userspace library about the subflows management: creation, deletion and traffic distribution.

The motivation for this project was to improve and unify the algorithms used in the multipath transports by providing the common policies manager in the userspace, which performs such tasks as address selection, name resolving and traffic distribution. For example, it could take links and payload properties into consideration and distribute traffic more cleverly according to applications requirements (payload type) and link properties (for instance, a link's RTT, the traffic cost or packets loss rate).

The result of this experiment was the prototype of interactive sockets that have a linked event descriptor to transfer events on a socket from kernel to the userspace. This investigation was useful to determine the direction in which multipath protocols could be improved in terms of the polyversal stack, for example unifying of distribution algorithms, better address selection and flows policies[110]. However, this experiment is specific for a linux kernel. For BSD kernels, it is required to add the same logic to the kqueue implementation (for example, by creating the new filter).

2.2.4.3 Conclusion

We have studied the state-of-art tendencies and ideas related to transport protocols and created the base prototype for the polyversal transport framework. The experiments carried demonstrated that ipc could be used as an effective transport between local applications, however it is required to take the peculiarities of the systems into consideration to provide optimal data transfer properties.

In-kernel prototypes, despite of their failed state in general, have demonstrated that the there are no real alternatives to the monolithic transport stacks in the kernel space. The existing systems, such as netgraph, are not suitable for creating transport framework without fundamental rework as they are not connection oriented and do not provide reliability nor state handling. However, the scheme where each data packet may have a set of labels used in MPLS networks could be used to create a framework suitable for implementing the transport layer and it is the subject of further investigations.

2.2.5 What Next?

The conjunction of a transport independent interface compatible with BSD sockets, transport hints managed by a DNS server and transport protocols graph is the novel idea that may improve communications over the Internet as well as in-host or intra-hypervisor data transfers. This is based on state-of-art ideas and researches, such as protocol independent API, and introduces the novel architecture of a scalable transport framework.

3 CPU Liquidity

This section describes mechanisms that we propose for achieving CPU liquidity. The main concept behind CPU liquidity is virtualization. Virtualized functions can be moved between different processors, effectively creating CPU liquidity. Early cloud computing pioneered the business of renting slices of computation resources in large datacenters to multiple (possibly competing) tenants. The basic enabling technology for this was operating system virtualization. Solutions such as Xen, KVM or VMWare provided a shared physical platform on which customers could host multiple virtual machines (VMs). Each VM presents itself as a self-contained computer, booting a standard OS kernel and running unmodified application processes, but with strong isolation from other VMs running on the same physical host. A key driver to the growth of cloud computing in the early days was server consolidation. Since existing physical installations of applications could be migrated into VMs wholesale, this made it feasible to switch off underutilized physical machines and pack their services into VMs instead. These VMs in turn could be backed up easily, live migrated to different physical hosts to balance shifting loads, and generally be cheaper to manage via software APIs rather than physical means. Cloud providers such as Amazon took on the burden of maintaining datacenters and keeping the physical machines powered and connected to the network.

The Trilogy2 project has been following two lines of work in the area of virtualization as a tool for CPU liquidity. First, efforts were conducted to support seemless migration of VMs on mobile hosts. Second, we have done work in the network function virtualization area. In particular, we worked on virtualizing the BRAS and CPE network components.

3.1 Virtual Machine Migration for Mobile Devices

3.1.1 Introduction

The goal of VM migration for mobile devices is to enable a unified user experience at the application level and to take advantage of the resources available in the nearby devices. Our approach is to transfer running applications and data between devices (desktop, Android tablets and phones) without changing the app state and while preserving its network connectivity. Thus, the user can change the hardware device while maintaining contact with the same application and available resources can be allocated to applications requiring them. The main target use cases can be divided into user (GUI) application migration and backend application migration. The first option consists of migrating applications like media players, VOIP clients, and email clients with the benefit of increasing the user's mobility and resource allocation. The backend application migration aims for better resource utilization. The solution is to encapsulate the application into a small-overhead container. This container is a VM that assures the proper link between the device's IO and a set of defined virtualized devices (eg. network, disk, camera, phone sensors etc.).

3.1.2 Virtual Machine Manager (VMM)

We decided to use qemu as a hypervisor for the virtual machines as it features hardware virtualization. Hardware virtualization enables efficient use of computing power and code is ran directly on the host computer's CPU allowing apps to have near native performance. There are two techniques for moving the app container from the source to the destination: pre-copy and post-copy [63, 64, 65] migration. In the pre-copy migration, the memory is copied to the destination, while the source container is still running. Before this process is finished, some memory pages may change and become "dirty". Those pages need to be re-sent. When a certain threshold is reached (number of iterations, size of dirty RAM) the machine is suspended on the sender, the rest of the ram is copied to the destination and the machine is resumed at the destination. The migration time depends on the number of rounds.

$$migTime = \frac{RAMsize}{NetworkSpeed} + \mu \tag{3.1}$$

μ depends on the number of rounds

In the post-copy migration, the VM is instantly suspended on the sender, and critical parts of the container are transferred (eg. the cpu registers and devices states). The virtual machine is then resumed at the destination, even though most of its memory is still at the source. When a page-fault occurs, memory pages are sent through the network. A separate thread sends the rest of the memory in background. This technique can increase the speed of the migration (as seen from the user's point of view) but can degrade the app performance onto the destination (in the first seconds – as the page faults are dealt and memory is transferred over the network). The migration time, as seen in formula 3.2, is constant, as no pages are being dirtied.

$$migTime = \frac{RAMsize}{NetworkSpeed} \tag{3.2}$$

3.1.3 Architecture

The proposed solution will be an application that has to be installed on the user's devices (mobile phones, linux PC). HTML5 apps will run inside containers, ensuring security and process isolation. Its main scope will be a virtual machine manager. Other features will be: per app permission, migration, auto-neighbor discovery. HTML5 has been chosen because it ensures portability. Any app developed using this architecture can use all the features that HTML5 has like capturing audio/video, accessing sensors, geolocation, touch events, offline storage etc. The proposed solution will include a view listing all the current running apps and stats about how much memory/storage they consume and CPU load. The user could modify the permissions of any running app, and could grant/disable any authorization while the app is running. If an app uses certain permissions (capture video using webcam) and the user removes it while it is running, a warning will be issued.

Another view will include a list of preferred devices to which apps can be migrated and an estimation of the

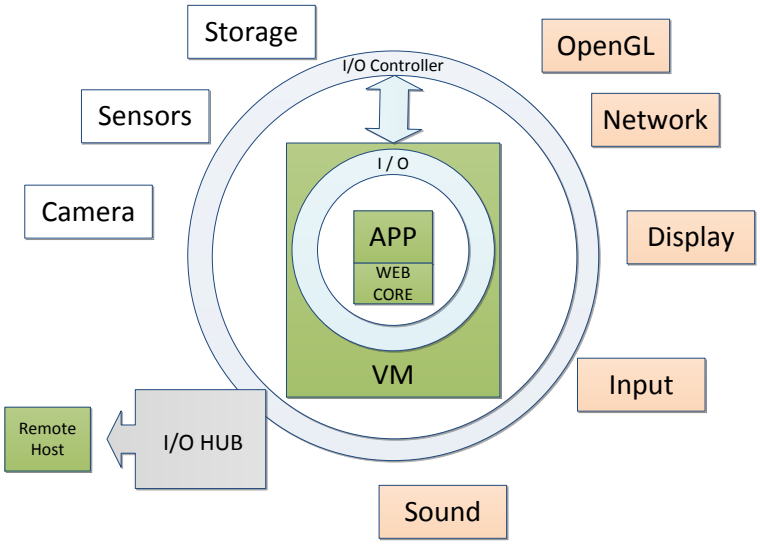


Figure 3.1: Liquidizing GUI applications – IO Manager Architecture

required migration time based on live measurements (latency, basic throughput test). Moreover, a separate list of devices will be available based on the auto-neighbor discovery. A user may choose to send his app to a third party device if he knows its IP address. When a user receives an app, a screen will appear listing the app's permissions (Eg: access the microphone), size, and estimated transfer time. He can choose to revoke some of the permissions and accept/reject the transfer. When the user closes the app manager, all the apps are suspended and their RAM is saved to disk. It will also warn the user if a certain app is consuming a lot of resources for a long period of time. The solution will only work on devices with hardware virtualization, so it can benefit near-native performance and efficient use of processing power. Moreover, OpenGL pass-through will be available to give the apps video performance.

3.1.4 Current Progress

Currently qemu is used as the hypervisor for the virtual machines. The Yocto[160] project is used to build a small footprint virtual machine (the app container). The machine contains the X server and a stripped down version of chromium. An I/O controller links the host's external devices with the guest's emulated devices. The I/O controller emulates generic devices so there are no compatibility issues when migrating apps between devices.

Live migration requires the VM disk to be replicated to both parties (sender and receiver) or to be present at a shared location. The chosen approach is to build a live operating system and put the ram and the disk together. The advantage is an easier migration, without the need to manage the disk. The limitation is that the app can consume a lot of memory and will not have enough disk space or if it is consuming all the disk space it won't have any more memory to allocate.

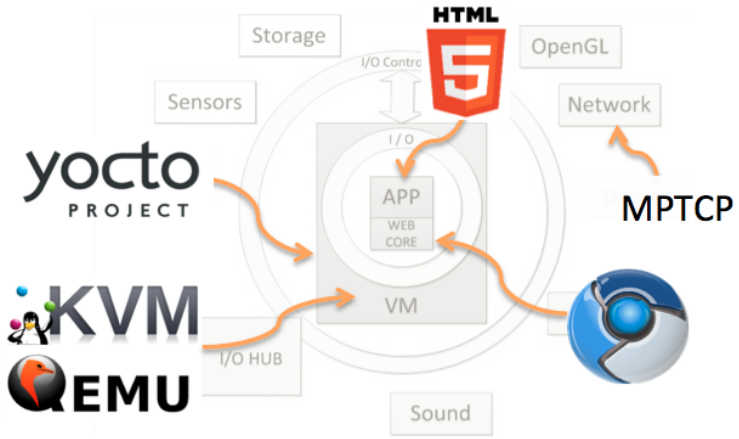


Figure 3.2: Liquidizing GUI applications – Implementation

3.2 High Performance Virtual Software BRASes

3.2.1 Introduction

One of the most crucial but lesser known components of today's networks is the Broadband Remote Access Server (BRAS), also referred to as a Broadband Network Gateway (BNG) [137]. BRASes are important: all access network traffic (e.g., DSL traffic) goes through them and they are in charge of a wide range of functionality critical to network operators. Among others, they take care of authenticating users, performing accounting and monitoring, handling sessions and tunnels, and shaping bandwidth, not to mention acting as the first IP point in the network.

Given their position in the network and all of the functionality they contain, it comes perhaps as no surprise that they are expensive, hardware-based proprietary boxes. Besides its price tag, a BRAS comes with a number of limitations. First, it presents a massive single-point of failure: one BRAS usually handles thousands of subscribers, something that can equate to a large number of customer support phone calls (and their related costs) should things go awry. This means that operators are loath to introduce new functionality or services in BRASes, lest they cause serious disruption to their operational networks.

Second, the fact that all session tunnels (e.g., PPPoE) are terminated at the BRAS means that any IP-based services such as multicast IPTV or access to CDN caches must sit behind the BRAS. This prevents operators from being able to place such services closer to end users (e.g., in the aggregation network), causing inefficiencies in the network and poorer service for users.

Finally, because they are hardware-based boxes, upgrading their functionality is less than trivial, often requiring a long wait, often years, until the next version is available; this prevents operators from deploying new customer services in a timely fashion. Also, BRASes are far from modular: operators must pay for all of the functionality even though they may only be interested in deploying some of it (e.g., PPPoE tunneling but not IGMP support).

Ideally, in order to tackle these problems, we would like to turn these rigid, monolithic hardware devices

into software-based network functionality that can be easily decomposed and dynamically instantiated at different points in the network. This concept follows the existing trend captured by the NFV approach [136] which seeks to turn hardware devices like middleboxes (e.g., NATs, DPIs, proxies, firewalls) into virtualized software-based processing running on commodity hardware (i.e., inexpensive, off-the-rack x86 servers).

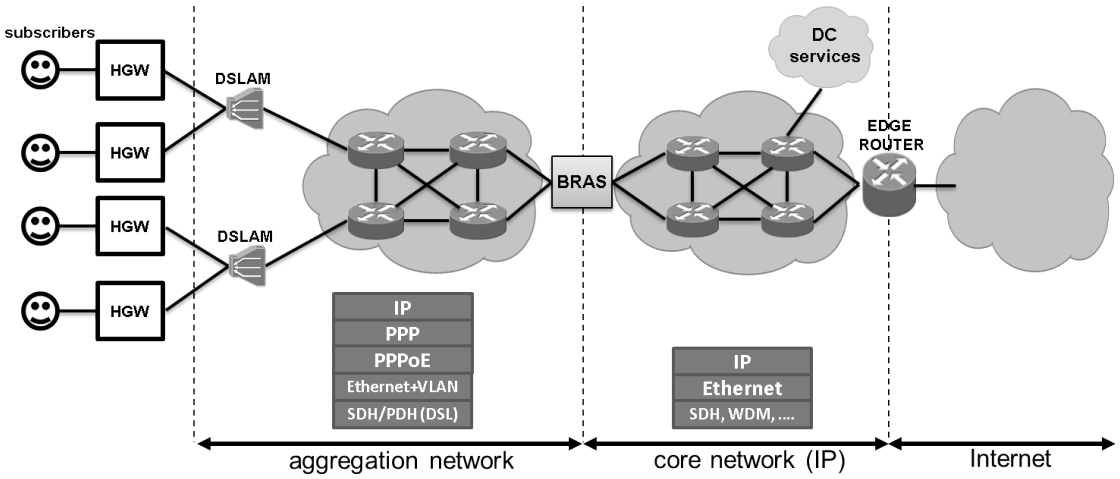


Figure 3.3: Broadband access network architecture along with related protocol stacks.

We present a proof-of-concept implementation and performance evaluation of such a BRAS. Our software BRAS is built on top of ClickOS, a lightweight Xen virtual machine (VM) based on the Click modular router software [79] and aimed at network processing. ClickOS VMs can be instantiated in tens of milliseconds, have a small memory footprint of 5MB when running and can process packets at 10Gb/s. In addition, we show that our software BRAS can establish PPPoE sessions at rates of 1000/s (the maximum that the IXIA tester we used could offer) while using a small amount of memory for the session state and the VM itself (about 20MB).

This software BRAS provides a number of advantages. Because it is software based, it is simple to upgrade, and Click's modularity means that adding functionality to it is also a quick affair (we will give an example of this in section 3.2.3). This modularity, and that afforded by the lightweight ClickOS VMs, allow for easy decomposition of the functionality contained in a traditional BRAS. As a result, we could envision deploying some functionality closer to end users, for instance to (1) break off tunnels at aggregation networks in order to provide services such as access to CDNs more efficiently; to (2) reduce tunnel overheads by allowing home gateways to send IP traffic without tunnels while introducing the necessary ACLs to control such traffic; or to (3) leverage the availability of large numbers of home gateways, many of which contain general-purpose processors, in order to scale processing.

In addition, the fact that the software BRAS is virtualized means that operators can test out new features on a small subset of subscribers without risking adverse effects on subscribers being handled by other VMs, perhaps removing a barrier to innovation. Finally, the software BRAS is relatively inexpensive: the server we ran our tests on costs in the region of $2,000.

In section 3.2.2 we introduce background information regarding access networks, protocols and BRASes

that will help understand the details of our implementation. Section 3.2.3 describes our implementation and presents a performance evaluation of the software BRAS. Finally, section 3.2.4 discusses, in greater detail, the scenarios that are enabled by a virtualized software BRAS and section 3.2.6 offers some concluding remarks on this topic. This section is based on work presented at the Second European Workshop on Software Defined Networks (EWSDN) in October 2013.

3.2.2 Background

In this section we give a brief background about broadband access networks and BRASes; this will make it easier to understand the implementation of our software BRAS in the coming section.

3.2.2.1 Broadband Access Networks

The most common technologies used by network operators in the last mile of the network are Digital Subscriber Line (DSL) and cable, with DSL being the most deployed technology, especially in Europe [138]. In a typical set-up, users' home gateways (e.g., xDSL routers) are aggregated at a Digital Subscriber Line Access Multiplexer (DSLAM). From there, the connections going through several DSLAMs are themselves combined using an aggregation network whose end point is the BRAS (see figure 3.3). The BRAS acts as the first IP point in the network, and connects to the operator's core IP network, which may itself be directly connected to data centers to provide localized services to its customers. Finally, one or more edge routers provide connectivity to the Internet.

A network operator provides connectivity to its subscribers by binding the subscriber to the concept of a session: the subscriber has to establish a session to get access to the network and its services. The session establishment usually involves procedures for the authentication of the subscriber and the application of network policies (e.g., bandwidth limits and quotas), and is initiated by the home gateway (HGW) and terminated at the BRAS. The most common protocol used for this purpose is the Point-to-Point Protocol (PPP), together with PPPoE, the variant used to work over Ethernet [137]. This effectively creates a tunnel where all the network packets sent or received by the HGW must be encapsulated in PPPoE/PPP headers and decapsulated by the BRAS. Figure 3.3 shows the relevant protocols stacks at each point of the access network.

3.2.2.2 Broadband Remote Access Server

As previously mentioned, the BRAS carries out a wide range of tasks. These include, but are not limited to:

- PPP/PPPoE session management and header decapsulation.
- Interface to AAA (Authorization, Authentication and Accounting) services.
- Traffic shaping, generally in the form of a hierarchical shaping policy that creates per subscriber and per service classes.
- Access control based on access control lists (ACLs), making sure that packets belong to a subscriber for whom a session has been already established.

- ARP proxy, replies to ARP requests coming from interfaces on the BRAS' IP side.
- IP forwarding to routers in the operator's core network.
- Assignment of IP addresses to subscribers.
- IGMP processing in order to support multicast.

In the next section we will describe our software BRAS implementation, including how its modularity allows us to decompose the large list of functionality contained within a traditional BRAS.

3.2.3 Proof-of-Concept

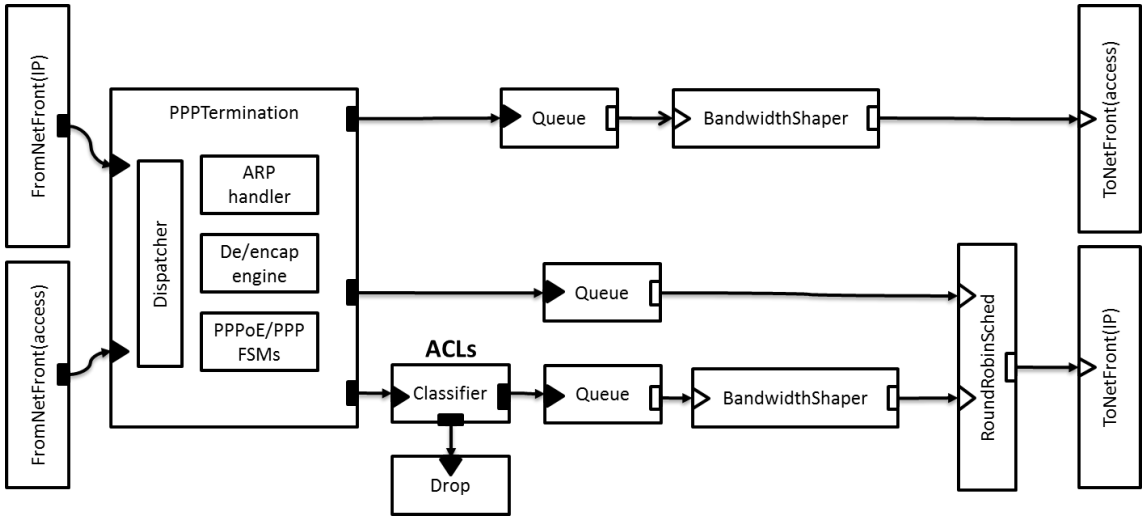


Figure 3.4: Software BRAS design.

We will now describe our proof-of-concept software BRAS implementation. We begin by giving a small introduction to the Click modular software. We then discuss ClickOS, the virtualization system that our software BRAS is based on, followed by an explanation of the BRAS itself. We end the section by presenting results from a performance evaluation of it.

3.2.3.1 Mini Click Primer

Click is based around the concept of modules called *elements*, each performing a small amount of network-related packet processing such as decreasing the TTL or re-writing packet headers. An element also has read or write *handlers* that let users inspect or change the state of the element (e.g., the AverageCounter element has a read handler to get the number of packets seen so far and a write handler to reset that count). Elements can be connected to other elements, resulting in a directed graph called a *configuration*. Thanks to its modularity and the fact that it comes with hundreds of elements, it is quite simple to use Click to come up with configurations representing IP routers, NATs, firewalls, proxies, and many other kind of network devices. Click is also extensible, making it easy to connect existing elements to new, custom-designed ones; this is the case for our BRAS implementation described further below.

3.2.3.2 ClickOS

ClickOS consists of lightweight, Xen-based virtual machines [14] (also known as guests or domUs). Xen has the concept of *paravirtualized* guests, whereby the operating systems of the VMs are slightly modified to hook into the Xen hypervisor and I/O subsystem; this is done in order to boost the performance of the guests. While mainstream operating systems such as Linux or FreeBSD have been paravirtualized and can thus run as guests in Xen, in this work we are only interested in doing *network* processing, for which a full fledged OS is overkill. What we would like is to have a paravirtualized, minimalistic OS to which we could add the necessary network functionality, and the Xen sources come with exactly the right thing: MiniOS.

MiniOS provides the basic facilities to run as a Xen guest with few overheads. For example, it has a very simple co-operative scheduler, so it does not incur context switch costs. In addition, it has a single address space, so there are no expensive system calls to speak of [1]. ClickOS, then, consists of combining MiniOS with the Click modular router software into a single Xen virtual machine.

What about networking? Xen implements a so-called split driver model, where a back-end driver called *netback* communicates with the NIC's device driver (usually via a software switch such as Open vSwitch [143]), and a front-end driver called *netfront* runs in the guest OS and talks to the netback driver using a ring API; one of the steps in paravirtualizing an OS is providing such a netfront driver. Xen networking does come with performance issues, so we had to heavily modify chunks of this subsystem in order to improve the performance of the software BRAS; details of these changes can be found in [84]. Finally, we added code to Click so that (1) it can interact with a netfront driver as opposed to a device driver and (2) it can run in a MiniOS environment which, among other things, does not provide a console to perform actions like installing Click configurations.

3.2.3.3 Software BRAS Design

The design of our software BRAS consists of a Click configuration centered around a new element we implemented called *PPPTermination* (see figure 3.4). The BRAS has two main interfaces, one facing the access network and another one connected to the IP network (four elements in total, one receive and one send element per interface).

Packets arriving from the access network go directly to the *PPPTermination* element which takes care of establishing sessions and handling tunnel decapsulation (more on this later). Packets for which a session exists are decapsulated and sent out of the *PPPTermination* element. From there they go on to a classifier that implements access control lists (ACLs) followed by a bandwidth shaper and finally out to the IP network.

On the reverse path, that is, packets coming from the IP network, the *PPPTermination* element handles ARP queries and replies. In addition, it takes care of forwarding IP packets onto subscribers by encapsulating them in PPP/PPPoE tunnels. From there packets go once again through a bandwidth shaper and then to the access network.

[1] Note that these do not make ClickOS less secure: in general, the model is that each ClickOS VM is owned by a single entity, and entities are isolated by the fact that they use different VMs.

The *PPPTermination* element deserves a closer look (figure 3.5). The element implements the PPPoE and PPP protocols (RFC2516, RFC1661, RFC1332, RFC1877). Both the PPPoE and PPP RFCs define a Finite State Machine (FSM) whose transitions depend on events generated by both network packets and timers. The FSMs are mainly involved during the set-up and tear down phases of a subscriber session and use a common database to store session information. The same database is then used by the ARP handler and by the encap/decap component.

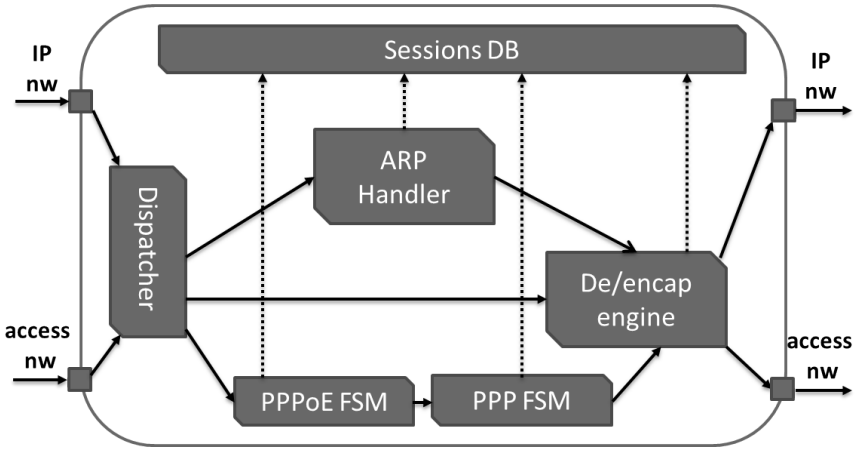


Figure 3.5: PPPTermination element design.

In greater detail, packets coming into the *PPPTermination* element first hit a dispatcher which is in charge of splitting ARP packets from IP ones (for packets coming from the IP network) and packets from established sessions from those seeking to establish sessions (for packets coming from the access network). For established sessions, the dispatcher simply checks whether a session ID exists and if so sends them onto the encap/decap engine which, in turn, verifies that the ID is valid, ends the PPP/PPPoE tunnel, and sends the packet out onto the IP network (this is the BRAS' fast path). Packets which do not have a session ID are sent to the FSMs in order to establish a session.

3.2.3.4 Evaluation

This section presents a performance evaluation of our software BRAS and the ClickOS system it is based on. The BRAS ran on an HP DL380G7 server equipped with two Intel L5640 (6-Core 2.26GHz) processors, 24GB of RDIMM PC3-10600R-9 memory and an Intel 10Gb card (82599 chipset). For traffic generation we used a similar server, and for PPPoE/PPP session establishment IXIA and Spirent testers [2]; all of these were connected to the software BRAS with a direct cable.

In addition, the Xen hypervisor was configured to assign four cores to the driver domain that hosts the device and netback drivers, while the remaining cores are used by the running ClickOS instances. Moreover, the driver domain is configured to use 1024 MB of system memory, with memory ballooning disabled.

Instantiation Times On top of the modularity provided by the software BRAS, it would be ideal if we could

[2]We used two different testers to ensure that our prototype was standards-compliant.

instantiate its functionality in different parts of the network dynamically and as quickly as possible to adapt to changing network loads, to deploy new services, or to apply new policies to subsets of subscribers.

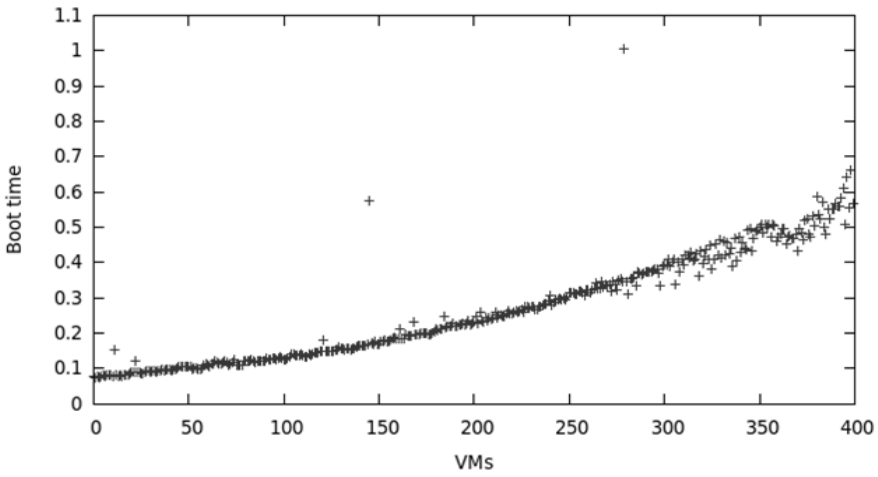


Figure 3.6: ClickOS boot times, in milliseconds.

As it turns out, the BRAS, running on top of ClickOS, can be booted extremely quickly, in as little as 70 milliseconds (figure 3.6). The figure also shows that a large number of ClickOS virtual machines can be quickly booted on the same, inexpensive commodity server. Note that this does not mean that we would be able to run hundreds of concurrent software BRAS instances while handling traffic, but gives an idea of how long it would take to instantiate BRAS functionality on a server.

Session Establishment To measure the session establishment rate (in sessions per second), we used a single ClickOS virtual machine running the software BRAS configuration. We performed several tests, each time changing the number of sessions generated per second up to the maximum of 1,000 that the IXIA tester could handle. All the tests are performed without using external AAA services in order to focus only on the performance of the BRAS.

The results, plotted in figure 3.7, show that our prototype is able to achieve a session establishment rate almost equal to the session generation rate. Note that typical commercial BRASes are able to establish about 300 PPP sessions per second on average (albeit while handling data plane traffic, which we did not do in the test above). Further, we performed the same tests using up to 26 concurrent software BRAS virtual machines and obtained the same results.

One of the big advantages of using lightweight ClickOS virtual machines is the small memory requirement: a compressed software BRAS virtual machine is less than 2MB in size. We performed a test to evaluate the amount of memory needed when establishing sessions. The results (figure 3.8) show that the memory consumption increases linearly with the number of sessions. More interestingly, only about 1MB of memory is needed per 1,000 established sessions, with just 8MB extra for the virtual machine itself. According to this test, a single BRAS instance needs less then 100MB of memory to accommodate the maximum number of sessions supported by the PPPoE protocol (65,536).

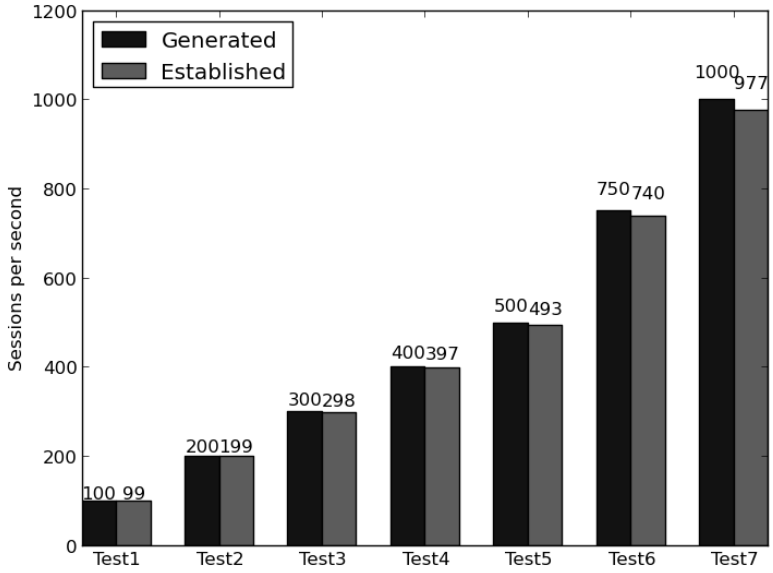


Figure 3.7: Software BRAS session establishment rate.

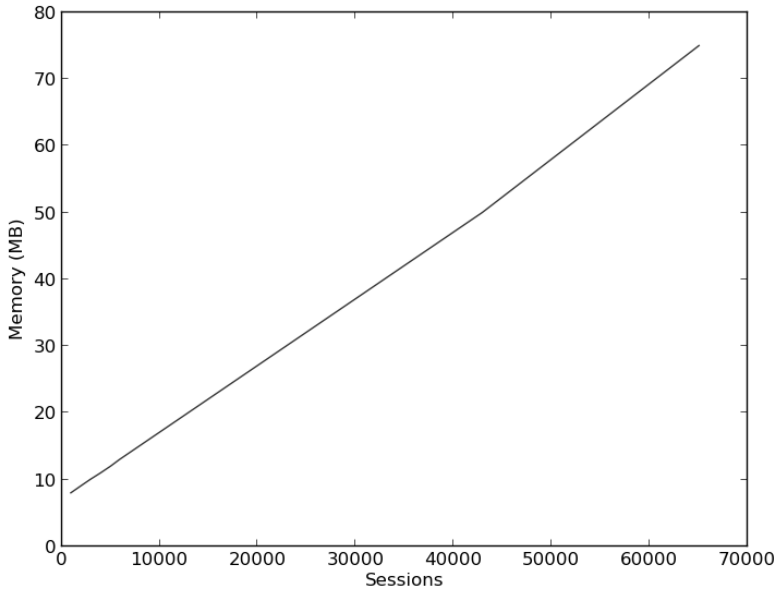


Figure 3.8: Software BRAS memory consumption when establishing sessions.

Throughput So far we presented results for boot times and session establishment, but what about the performance when handling large quantities of traffic? To evaluate this, we instantiate an increasing number of concurrent ClickOS virtual machines on the same server, instructing each to send out large rates of traffic for different packet sizes. All packets then converge on a single 10Gb port, and we measure the *cumulative* rate at a separate box connected via a direct cable.

We plot the results in figure 3.9. The flat lines mean that the system is able to sustain the same rate for each packet size despite more and more virtual machines coming online. It is also worth pointing out that ClickOS can fill up a 10Gb pipe for almost all packet sizes. For loads with only 64-byte packets (an aberrant case that

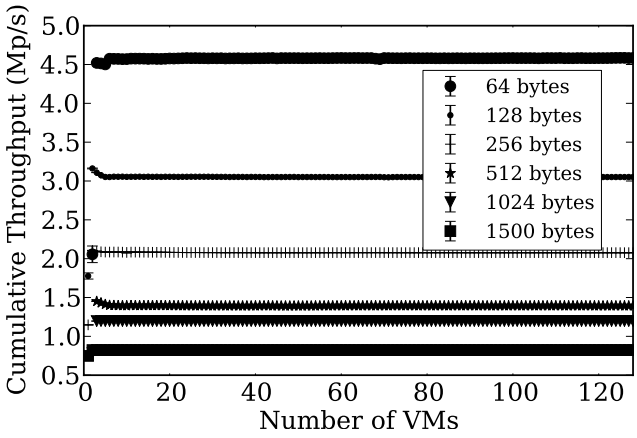


Figure 3.9: ClickOS throughput performance while concurrently running different numbers of virtual machines.

never happens in operational networks) it is still able to produce over 4.5 million packets per second. It is worth pointing out that the configuration used for these tests was a simple one rather than the software BRAS one. We are now in the process of performing similar tests when running the BRAS configuration.

3.2.4 Discussion

The early experimental results from our prototype are encouraging and make us optimistic about the applicability of a virtualized software BRAS in future broadband access networks. Our software BRAS solves many of the issues that current BRASes suffer from: it can be deployed on inexpensive general-purpose servers, it is non-proprietary, it allows for easy upgrades, and the fact that it is modular means that operators can pick and choose which functionality to actually deploy (and pay for).

In addition to these benefits, we believe that software BRASes like our prototype can enable a number of interesting scenarios that are difficult or impossible to achieve in current access networks. First, our BRAS allows for fast deployment of new functions by making it easier to conduct field tests: a pilot trial can be performed on a small subset of subscribers before deploying the solution on a larger scale, or rolled back to the operational implementation in case things fail. What's more, such a trial could be conducted on the same commodity server that is handling operational traffic without fear of disruption, since the trial would be running on a separate, isolated virtual machine.

Second, the ability to deploy different BRAS instances allows for the creation of targeted differentiated services aimed at users with different needs. For example, we could imagine a BRAS instance which supports virtual private networks for enterprise customers and another BRAS without such a function used for home subscribers, both running on the same inexpensive server. Indeed, the fact that our prototype is able to potentially run hundreds of instances on a single server means that we can create fine-granularity classes of subscribers, each with different policies.

Finally, the decomposition of the BRAS into its elementary functions, coupled with ClickOS' features like fast boot times and small memory requirements, allow for dynamic deployment of BRAS functionality on-demand and at different points in the access network, enabling further scenarios. For instance, the decap-

sulation/encapsulation function could be moved closer to the user to allow early tunnel termination. In this way, operators could give access to cloud and CDN services deployed in the aggregation network, a gain in efficiency compared to today's traffic which has to traverse the aggregation network, go through the BRAS, and only then access the needed services.

In the extreme, we could envision putting some of the BRAS functionality in home gateways. Their large numbers could be leveraged to scale the performance of the network further, and since many of them also tend to use general-purpose processors (often ARM ones), it would be possible to extend our prototype to them. For example, we could have a software BRAS instance handling session management install ACLs directly on home gateways, removing the need to carry traffic in PPP/PPPoE tunnels and thus getting rid of overheads in the access network.

3.2.5 Related Work

There is growing interest both in the research community and industry in turning traditional network processing as embodied by hardware devices like middleboxes into software-based functions running on cheap, commodity hardware. The authors in [129], for instance, advocate for turning middleboxes into software functions and deploying them as a service in the cloud in order to reduce management complexity and costs. Earlier research looked into scaling software router performance by constructing a single router out of several commodity servers [36]. In contrast, the work presented here does so in a virtualized environment, is able to fill up 10Gb pipes with a single inexpensive box, and focuses on implementing a BRAS instead of a router. The authors in [38] looked into measuring the performance of virtualized software routers, though that work was aimed at 1Gb interfaces and routing only.

On the industry side, a large number of major operators from around the world have recently formed the ETSI Network Functions Virtualization Industry Specification Group in order to define and try to solve the major challenges in turning hardware devices into software-based functions [72]. Regarding vendors, Cisco sells a virtualized router than can run on VMware ESXi or Citrix XenServer [30]. Vyatta [144] provides open-source software that implements middlebox functionality and can run on different virtualization platforms. Our work focuses on BRASes rather than middleboxes, uses lightweight virtual machines that can boot quickly and have a small memory footprint, and provides a high-performance virtualized environment.

3.2.6 Conclusions and Future Work

We have shown that our prototype can handle large session establishment rates and can instantiate BRAS functionality very quickly. These properties, along with the prototype's modularity, make it much easier for BRASes to be updated and new features tried out without fear of disrupting service to the majority of subscribers. In addition, we discussed a number of scenarios that our software BRAS enables, thanks to the fact that it can be easily decomposed and dynamically instantiated in different parts of the network.

As future work we are planning to test the performance of the ClickOS virtual machine's packet processing rate while running our software BRAS configuration. Further, we are working on mechanisms to be able to

easily migrate session state between BRAS instances for reliability reasons or to to be able to dynamically scale out processing from an overloaded software BRAS. Finally, while the software BRAS configuration is modular, we are planning on doing performance tests where the BRAS functionality is carried out on different servers in order to improve scalability (e.g., one server handles session establishment, another one ACLs, and yet another one traffic shaping).

3.3 Virtual CPE

Current datacenter infrastructures are facing the rapid development of the cloud market, which includes a broad range of emerging services and distributed applications. Next generation datacenters are required to maximize bandwidth capacity and throughput to cope with the increasing performance demands bound to heterogeneous and ultra high bandwidth datacenter traffic flows generated by cloud computing applications in virtualized environments. Moreover, future datacenters need to become dynamic infrastructures with flexible network architectures capable to scale up and down while optimizing performances and resources utilization

3.3.1 Problem statement

Network providers that operate IP/MPLS networks usually sell Layer 3 Virtual Private Network (VPN) services to customers to let them interconnect their physical sites. Figure 3.10 shows an example where a network operator (called "X") offers to its customers basic IP/MPLS VPN services to extend their Layer 3 connectivity beyond each site boundary and interconnect remote sites. With the recent growth of cloud-based services, network operators are going to extend their pure network-based product portfolio (e.g. composed by Layer 2 and Layer 3 virtual connectivity services) with additional Infrastructure as a Service (IaaS): to do that, it is common that network operators build (or even rent) their own datacenters, usually directly connected to their networks.

3.3.2 Reference scenario

The telco market is moving towards "everything as a service", and the dynamic and on-demand integration of classic IP/MPLS VPN services with advanced IaaS can have significant impact on the network operators' businesses. In this context. the goal of the virtual CPE in the network operator datacenter is to combine virtualized IP/MPLS VPN services with IaaS for business customers, enabling network operators to provide a liquid VPN service with a beneficial impact on their resource pool usage, improving the flexibility at both datacenter and network levels. Figure 3.11 depicts the virtual CPE scenario: a business customer that interconnects its sites through IP/MPLS VPNs services offered by the network operator buys a set of IaaS resources (e.g. VMs, storage areas, etc.) from the same operator and want to extend these Layer 3 VPNs to incorporate the new virtual assets into a private cloud. In this context the virtual CPE can be seen as an additional control VM deployed on-demand by the network operator in the datacenter to dynamically join (or release, when needed) the customer Layer 3 VPN memberships, i.e. configure the proper Virtual Routing and Forwarding (VRF) instances at the datacenter edge router (Provider Edge (PE) in Figure 3.11). Therefore,

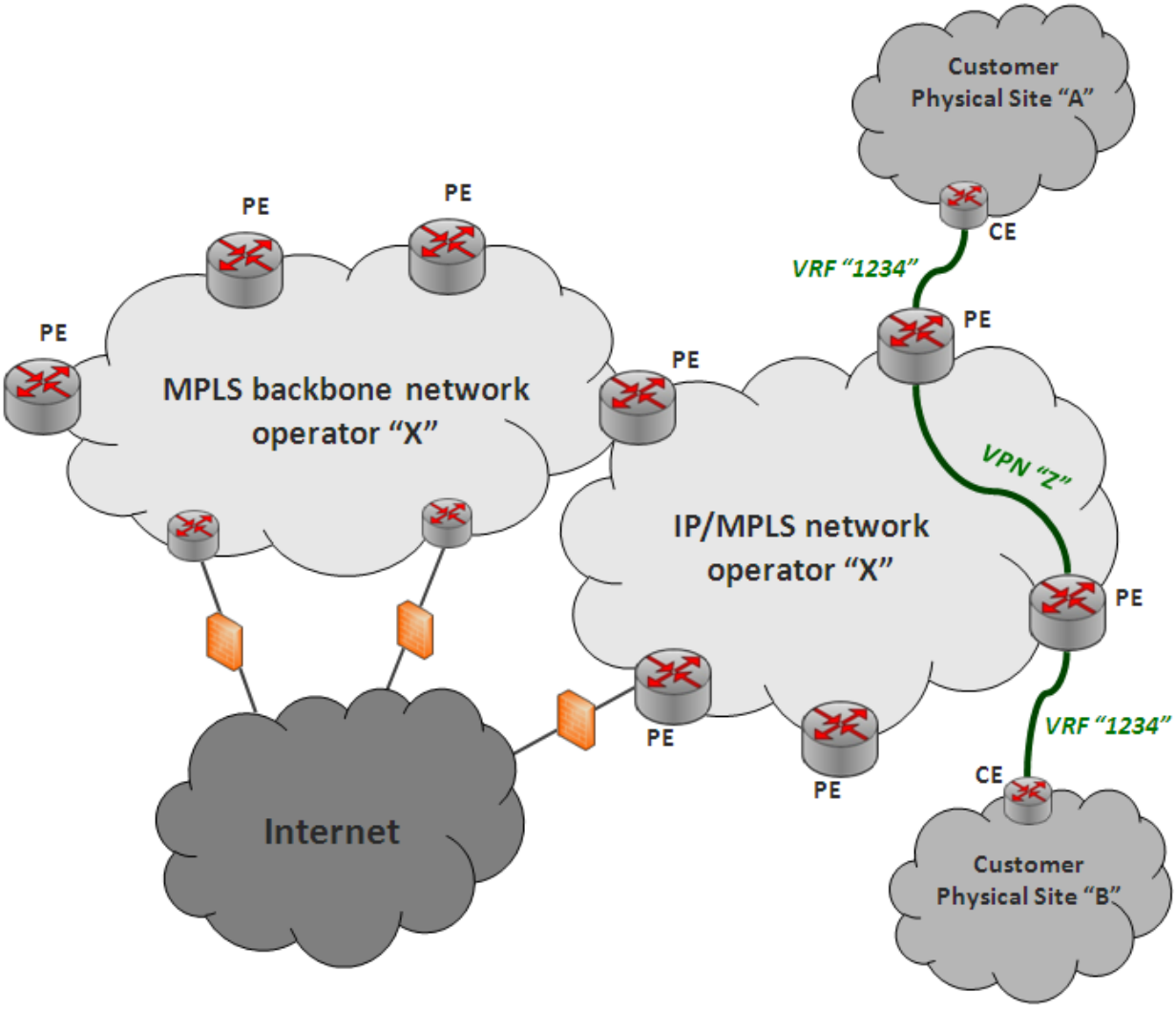


Figure 3.10: IP/MPLS network operator basic VPN services

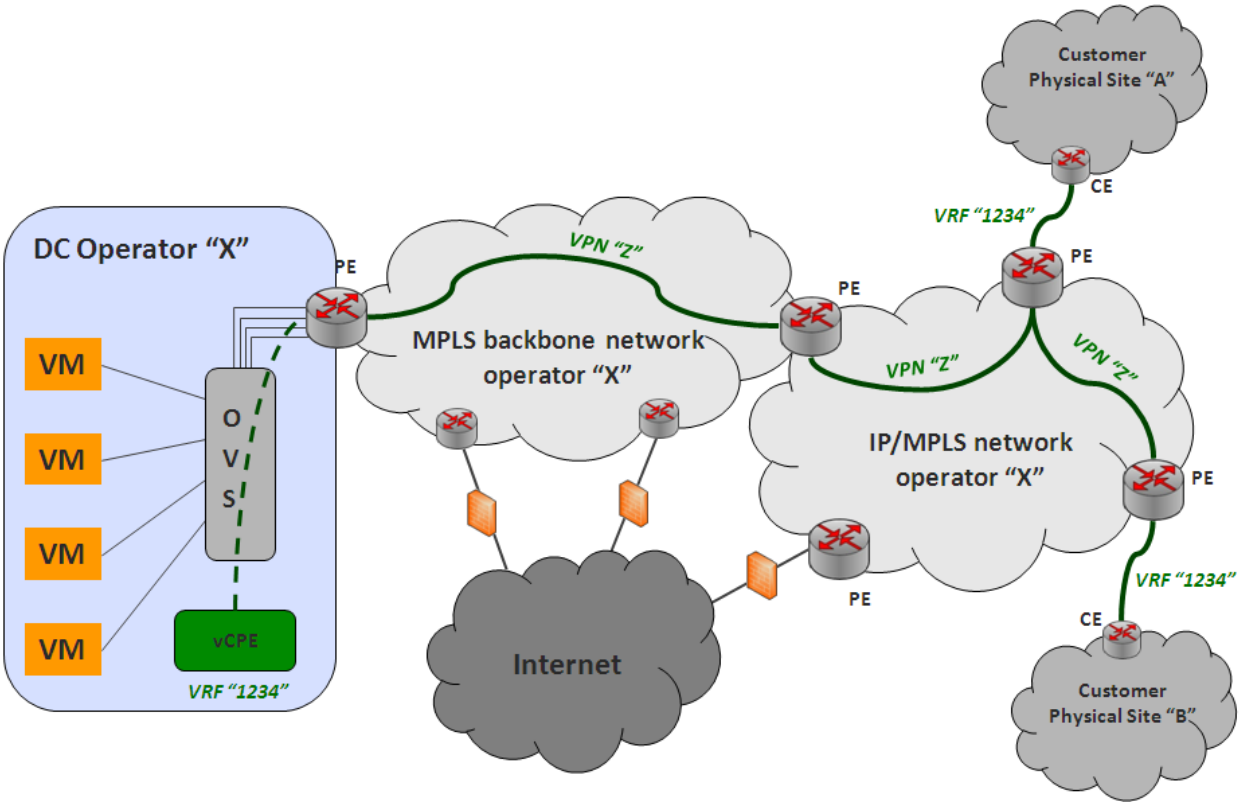


Figure 3.11: Virtual CPE in the datacenter scenario

the virtual CPE enables a seamless extension of IP/MPLS VPNs to the datacenters, and allows the network operator to provide enhanced cloud-based services over its VPN networks.

The scope of the virtual CPE, in terms of operational roles and responsibilities, is limited to the only datacenter segment; indeed, the aim is not to touch the IP/MPLS backbone configuration (in terms of VPNs, LSPs, etc.) for these IaaS operations. Therefore, the customer VPNs must be configured in the IP/MPLS backbone (and also available at the datacenter PE router) before the virtual CPE deployment; this can be assumed at least for those business customers who will buy IaaS (and not for those who only buy basic network services from the operator).

Upon the request of a business customer to buy some IaaS resources and include them into a private cloud accessible through the IP/MPLS VPN services already in place, the virtual CPE is deployed on-demand as part of the IaaS to extend the customer's VPN. This can be done by running Open virtual Switchs (OVSs) in the datacenter hypervisors, that allows to perform VPNs and IaaS service chaining via software, enabling a flexible and on-demand Software Defined Networking (SDN)-based virtual CPE. This allows to exploit the benefits of SDN architectures [149], mainly in terms of programmability and virtualization of network functions.

3.3.3 Functional architecture

The virtual CPE functional decomposition is depicted in Figure 3.12, where the main functional entities are highlighted along with their interactions. The core part is represented by the OpenFlow enabled SDN

controller (one for the whole datacenter), that in combination with a set of enhanced network applications provides on-demand flow and routing tables configuration at the OVS and datacenter edge router. In addition, a Cloud Management System (CMS) like OpenStack [100] or CloudStack [31], owned by the network operator and controlling its datacenters (may be more than one) is in charge of the on-demand deployment of the IaaS resources (mainly VMs, storage, etc.) requested by the business customers. Most of the components depicted in Figure 3.12 are unique for the whole datacenter and the whole network operators' business customers: the CMS, the IT configurator, the SDN controller, the network applications. The virtual CPE component that is deployed on-demand for each new customer chaining of VPN and IaaS services is the External Border Gateway Protocol (BGP) VM: indeed, as shown in Figure 3.12 (dotted red box) the actual virtual CPE functions are mostly provided by the combination of this Virtual Machine (VM), the SDN controller (for the control part) and the OVS (for the data switching part).
A detailed description for each of the virtual CPE functional module is provided below, in terms of main functionalities, roles and responsibilities.

- **Cloud Management System:** It is a cloud oriented integrated platform used for provisioning of IaaS in the network operator datacenters. It is fully controlled and operated by the network operator, and it is responsible for the on-demand orchestration of IaaS resource deployment (mainly VMs) and configuration of virtual CPE functions inside the datacenter. It includes a dedicated internal function for IaaS resource deployment: the IT configurator. It is in charge of creating (e.g. from available templates), setting up and configuring VMs, storage resources, etc. as requested by the business customer. For the virtual CPE part, the CMS triggers the SDN controller network applications for Layer 3 VPNs configuration and membership. Some examples of possible CMS are: OpenStack, CloudStack, etc.

- **SDN Controller:** The SDN controller is mainly used to perform the control actions for all the OVSes deployed in the datacenters through the OpenFlow protocol [99]. It configures proper rules in the OVS flow tables with two main purposes. First, to isolate traffic of VMs belonging to different customers (e.g. through the use of VLANs) and associate them to the proper Layer 3 VPNs at the datacenter edge router. Moreover, it allows the External BGP routing protocol to peer with the BGP instance running on top of the IP/MPLS backbone PE router. The deployment of the SDN controller allows to implement a software driven virtual CPE, with flexible and programmable network functions for VPN and IaaS service chaining, that can either be deployed as embedded or external applications.

- **Virtual CPE Configurator:** The virtual CPE configurator is an SDN network application that can be implemented either as an embedded or external function of the SDN controller. It is triggered by the CMS for the control of the network part of IaaS and VPN service chaining, and its main scope is twofold. First, through the SDN controller, it performs the configuration of the virtual CPE routing daemons running in the virtual CPE VMs (i.e. the External BGP VM), mainly in terms of VRFs to be

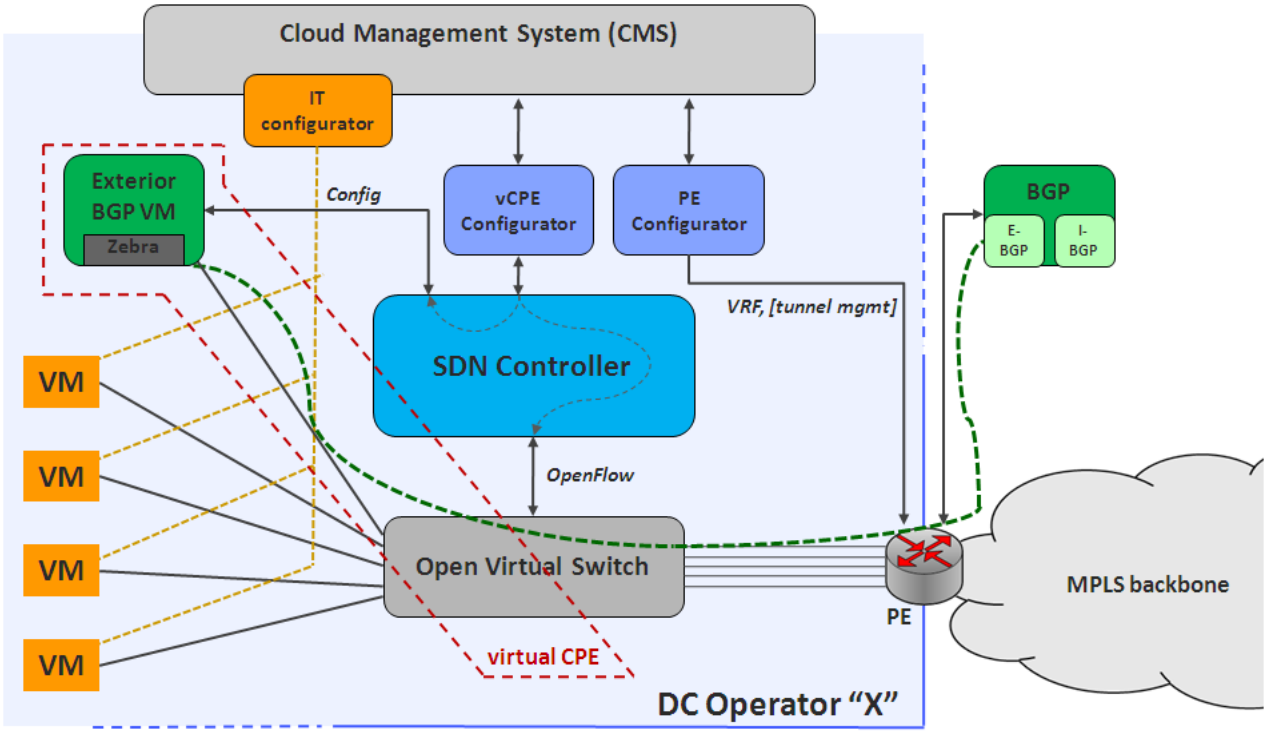


Figure 3.12: Virtual CPE functional decomposition

flooded. Moreover, again through the SDN controller, it sets flows and routing tables in the OVSes for both VMs and BGP protocol traffic purposes.

- **PE Configurator:** The PE configurator is another SDN network application, and as the virtual CPE configurator, it can run either embedded or external to the controller itself. The main role of the PE configurator is to set the VRFs in the datacenter edge router (i.e. the PE in Figure 3.12) upon the request coming from the CMS, and therefore let the traffic generated by the customer's VMs be associated to the proper VPNs. Optionally, the PE configurator may perform some additional MPLS management actions on the PE router to dynamically create or modify MPLS tunnels/LSPs to support new VRFs and establish new VPNs.

- **Exterior BGP VM:** The External BGP VM is the control VM allocated on-demand by the CMS to implement the main virtual CPE functions, such as dynamic join of Layer 3 VPN memberships and VRF information exchange with other BGP instances running in the IP/MPLS backbone. It mainly implements the BGP protocol, that allows IP/MPLS routers to exchange network reachability information and VRFs: in particular it runs the exterior BGP protocol to define the routing needed to let customer VMs traffic to pass across networks (i.e. from datacenter to customer sites, through backbone) and join the proper Layer 3 VPNs. A possible option to run the exterior BGP protocol in this virtual CPE VM is to use Quagga [113], a routing software suite that provides implementations of several routing protocols (including BGP-4) for Unix platforms.

3.3.4 Next steps

The plan for next activities on virtual CPE is to focus on the implementation of missing functional components: virtual CPE configurator, PE configurator and the BGP VM. The idea is to use OpenStack [100] as the CMS, which already provides the IT configurator functions, and one of OpenDaylight [98] or Floodlight [45] as SDN controllers due to their native support and integration with OpenStack. Concerning the on-demand BGP VM the plan is to develop in the Quagga framework the needed extensions to interact with MPLS routers, and also to leverage the work carried out in the project on control plane functions for liquidity.

4 Storage Liquidity

This section presents the work done by the Trilogy2 project in relation to storage liquidity. One specific resource that the partners of T2 are focusing on is storage. The main reason, though, is that the demand for storage is continuously growing. Analysts such as IDC have predicted that due to Big Data and the rise of user content on the internet that there will be a compound annual growth rate (CAGR) of 53% for storage between 2011 and 2016[1]. Revenue from storage is predicted by Ashish Nadkarni to increase from $380M in 2011 to $6Bn in 2016[2].

Of the three resource types (network, compute and storage), storage is the most tangible. The semantics of describing storage are well defined and there is one standards group that has captured most of the relevant concepts and is adopted by a large sector of the consumer and industrial market (for more info see Standards / SNIA in 'D2.2 - Information Model').

This section describes two mechanisms for storage liquidity. The first is Irminsule, a large-scale, immutable and branch-consistent storage solution. The second is Trevi, a radical new storage mechanism that uses fountain codes [82, 130] to provide resilience, high performance, and efficient resource utilization.

Other work for improving the exposure and control of storage resources is expected in the scope of Trilogy2 but as these are at different stages they have not been explicitly detailed for this Deliverable. A number of partners have been working on efforts into the Content Distribution Network (CDN) and storage content caching that can be used for optimization of the client experience, by reducing the latency to content and increasing the bandwidth. This relates network with storage resources and is something that will be investigated in more detail as the project progresses.

4.1 Irminsule

4.1.1 Introduction

Irminsule is designed to be the substrate for building self-managing and self-healing systems. *Self-managing* because the ability to checkpoint everything in independent branches lets a scheduler have a lock-free global view of activity that it can use for automatic resource scheduling across diverse system components. *Self-healing* because the provenance graph lets us rollback entire distributed clusters, taking advantage of immutability.

Sqlite (which can be embedded in any device that supports C compilation) has been embedded in billions of hosts worldwide (every mobile phone, browser and major operating system uses it somewhere). We aim for similar portability with Irminsule, but with the additional requirement of static type-safety for security and reliability.

We begin by picking three diverse deployment scenarios for Irminsule to emphasize its heterogenous nature.

[1] http://www.idc.com/getdoc.jsp?containerId=prUS24069113
[2] http://datastoragereport.com/big-data-drives-big-demand-for-storage

4.1.2 Distributed VM Storage

Virtual machine storage is usually handled at the block layer, with the VM layering its own filesystem over a block device. Irminsule operates at the *filesystem* layer instead, exposing a directory and metadata structure to the guest. This permits fine-grained control over how differences are stored, instead of having to map bidirectionally between the filesystem and block layers. This also makes offline manipulation of the VM contents significantly less complex (e.g. for online updates without shutting the VM down).
The initial design works via:

Irminsule Store is a git-like tree of the filesystem content and metadata, just as if an entire filesystem has been checked into git.

VM Working Tree is a conceptual checkout of a store revision into an area where the data can be freely mutated. This is exposed to the VM, and can be synchronized against the block store via ioctls to freeze the data into a staging area.

POSIX or NFS export maps a mutable working tree to POSIX or NFS semantics, using a scratch area for soft state such as lock information. It should be possible to turn this layer into a distributed lock manager for shared filesystem mounts, rather than complicating the remaining two.

We still need to think about how the Irminsule store maps this filesystem data to blocks. Ideally, blocks will be resilient to multiple host failure, and can easily be checksummed and replicated due to the immutable nature of the store.

4.1.3 Unikernel storage

The recent work on Unikernels has shown how we can eliminate the need for a POSIX layer for many applications, and this evaluation explores the benefits of using Irminsule as a *direct* block store without an intervening filesystem API. This is a much wider API than a POSIX interface in some respects, as it exposes the push/pull interface directly to the application. Locking has to be provided via library functions that use push/pull to expose more strongly consistent atoms.
Scheduling block updates is particularly interesting here, as we should be able to apply multipath-TCP-style congestion control to decide when to apply whole sequences of push/pulls (by using time-of-flight and spindle response times as replacements for TCP windows and congestion notification).

4.1.4 Heterogenous storage and coordination

Unikernels also open up the intriguing possibility of compiling directly to very heterogenous architectures. Irminsule is written in pure OCaml, and can be compiled into FreeBSD kernel modules, Javascript modules, and ARM native code for embedded systems. This allows us to use the immutable push/pull API as a coordination layer for very hetereogenous systems, by having them subscribe to pull streams from neighbouring nodes and committing their changes locally.

This model unifies fault-tolerance (an essential property for any embedded or web system), concurrency and distribution, at the cost of introducing storage garbage collection as a new issue (due to the immutability of Irminsule). Any evaluation strategy will have to take all this into account.

A great use case: Javascript applications can register API calls to remote services via local commits to an HTML5 local storage, and a background worker thread "flushes" these to a remote server that can commit them reliably. We have exactly this problem with Github API calls for workflow and commenting systems, and Irminsule offers an easy way to write web applications that can work both offline and online.

4.1.5 Design

Distributed databases usually make the distinction between managing concurrency coming from many different users and background administrative tasks trying to asynchronously managing consistency invariants. Each problems have received numerous contributions in the academic and industrial world, and the usual way to solve them is to use "optimistic" techniques, which make the assumption than the need for strong synchronization points on large-scale systems is usually minimal.

Irminsule adopts a more generic approach by giving to users and applications the same tools to manage both concurrency and replication. These tools are based on the notion of *branch-consistency* – dubbed like this as the fork/pull/push/merge life-cycle is similar to the one of distributed version control systems. In this model, application builder have to choose the type of data they want their application to work on and explicit how merge conflicts should be handle. In exchange, they got a loose consistency model, where ACID properties are guaranteed on each branch and where they have a direct control on the replication strategy.

4.1.5.1 Branch-Consistency

Designing distributed database is an hard problem, mainly because of CAP's theorem [55] which states that "It is impossible for a distributed computer system to simultaneously provide all three of the following guarantees: (i) *Consistency*: all nodes see the same data at the same time; (ii) *Availability*: a guarantee that every request receives a response about whether it was successful or failed; and (iii) *Partition tolerance*:(the system continues to operate despite arbitrary message loss or failure of part of the system."

The solution taken by most large-scale distributed system to solve the CAP's theorem paradox is to relax the consistency models and to admit, once for all, that there is no unique global state of the system. This class of systems are said to be *eventually consistent*. Purely eventual consistent system are very efficient but not very practical, as it is quite hard to guarantee the speed at which the consistency is propagated to the system. Thus, people have started to design mixed models, where the global system is eventually consistent, but user can subsystems with local strong consistency [32, 80] or wit strongly consistent transactions [24, 81, 135]. Unfortunately these models have many different formal semantics [22]) and remain hard to explain to the user.

An other interesting approach, that is followed by Irminsule, is to take inspiration from the distributed version control systems such as Git [150] and Mercurial and consider the *branching* operation as first-class: every

actor can create a new branch where the ACID properties hold, and detecting and merging conflicts between branches is an explicit operation defined by the user (we develop this idea in Section 4.1.5.4). We call such class of system as *branch-consistent* models.

This idea has starting to be explored on the programming language side by the *Concurrent Revisions* model [23] and we want to explore the design space of branch-consistent databases with Irminsule.

4.1.5.2 Optimistic Replication

Data replication [121] is a key technology in distributed systems that enables higher availability and performance. Usually people distinguish two kinds of replication: "pessimistic" methods, where we use master election and synchronous locking to block the system in while changes are propagated – these systems work well in local-area networks in which latencies are small and failures uncommon, but do not scale well. At the opposite, "optimistic" methods let the changes propagate in the background and have special techniques to handle occasional conflicts (hence the name *optimistic*: conflicts are supposed to be very rare).

Irminsule choose to use an optimistic replication system because (i) it improves availability: applications make progress even the network is unreliable; (ii) they are flexible because they do not need a-priori knowledge on the underlying network (they can use gossip and epidemic network replications); and (iii) they can scale to billions of nodes because they do not require synchronization. The drawback is that the users have to handle conflicts (see Section 4.1.5.4).

4.1.5.3 Publish / Subscribe

The optimistic replication strategy of Irminsule relies on a push-pull strategy. Such strategies has been studied for a long time in large-scale peer-to-peer systems [41] and it is a good way to decouple space, time and synchronizations: actors do not need to know each other to participate to such interactions (you just need public event services); the interactions do not need to happen at the same time and publishers are not blocked while producing events, while subscribers can get asynchronous notifications of events.

The global replication strategy is let to the user as library building blocks, choosing either a stochastic strategy [142] to elect pair of actors to synchronize in the background or, in order to better shape the traffic, a tree-like structure with low-level multicast (using congestion control techniques coming from the other T2 partners and the fountain coding low-level multicast from George).

For the concurrency model part, a push/pull strategy is also useful to keep track of provenance, which can lead the merging strategies in case of conflicts. In Irminsule, we have decided to synchronize states (instead of synchronizing sequence of operations as it is usually the case in other distributed databases) and we use the fact the underlying system is immutable and the computation of keys is deterministic (see Section 4.1.6) to optimize the amount of data exchanged between actors.

4.1.5.4 Conflicts

As we said earlier, conflicts can appear in two different areas: (i) two nearby users are modifying the same value at the same time (we call this concurrency conflict); and (ii) a value has been changed in two distant

locations and a conflict happens as the result of a background propagation (we call this a replication conflict). The philosophy of Irminsule is to give the ability to the application builder to deal with both kind of conflicts with the collection of tools. Among them:

- Conflict-Free Replicated Data Types [128] (counters, sets, multi-values store, etc).
- Type of data with custom merge operator. The merge operator can be tailored to work for trees [109, 57] or on more generic "cloud" operations [21].
- Every sequence of operations between branch and merge can take as well a callback function which is called every time a conflict happen, which can take some sort of context to help the resolution programatically – in a way very similar to transactions.

4.1.6 Architecture

In order to implement the design principles in an efficient way, Irminsule design consists in three main components: a *low-level* immutable and consistent key/value data-store (Section 4.1.6.1), a DAG persisted in that datastore (Section 4.1.6.2) and a *tag* store which associate names to keys of the low-level data-store (Section 4.1.6.5).

4.1.6.1 Low-level Data-store

At the core of Irminsule lies a very simple key/value data-store (the *low-level* data-store), where keys are names computed automatically from the values, with no user control. This means that:

- if you modify a value, its computed key changes as well: hence you are creating a new key/value pair. Hence the resulting data-store is *immutable* (eg. append-only); and
- if two data-stores share the same values, they will have the same keys: the data-store is *consistent*: the overall structure of the data-store only depend on the stored data and not on any external user choices.

The usual deterministic function to compute the key from a value is SHA1, which is subject to collisions (and with known collision attacks). Two possible answers:

- The *Git* way: we always preserve the existing data in case of collision, which means that the existing data is never corrupted.
- The secret seed: keys can have a secret part, generated by the creator of the value. We can then have, in background, threads rewriting the values to merge common values (and keep separate different values with the same hash).

4.1.6.2 High-level Data-structures

Irminsule uses the low-level data-store to implicitly encode and maintain a more complex data-structure, which can be separated into two complementary layers: one to encode a hierarchy of values (which can be seen as a filesystem) and one to encode the modification history as a partial order of snapshots.

4.1.6.3 Hierarchies

The first higher-level layer is a tree-like data-structure to model a *hierarchy* of nodes, with two kinds of nodes: value nodes, which contain raw values ; and directories nodes which contain the list of named nodes (which can be seen as sub-directories and files). Depending on the application needs (eg. on what the application builder has decided), the values can be structured values (such as counters or sets), raw binary files or block devices. As we said earlier, the user should define the associated merge function for each type of values he decides to use. This can have an huge impact on performance: for instance, choosing a raw file instead of a list of blocks for final values make the write operations much more costly as you have to allocate a new file.

A full hierarchy can then be trivially associated to the key of its root node, as long as we make sure that the order of in which the children keys are enumerated is consistent: recall that the key of a node is computed from its contents, so we want to be sure that two identical directories structures (possibly on different Irminsule instances) have the same key.

As the tree structure is encoded in the nodes themselves and the data-store is immutable, modifying a value in the hierachy means that the blob will change, and thus all its parent directory until the root needs to change as well. This means that updating any value in such a tree will result in a new tree, with most of the nodes shared with the original tree but all the path from the root to the modified value. In Figure 4.1, the original tree has the node n_1 as root. This tree contains three values: one located at `a` containing the value `Foo`, one located at `b/x` containing `Hello` and one located at `b/y` containing `World`. If we want to change the contents of `b/x` from `Hello` to `Bye`, we create a new tree, whose root is n_6 that share its node with the original tree but the node going from the root to the changed value (eg. if shares n_5 and n_4 but n_1, n_7 and n_8 have been created, depicted with dotted lines on the figure).

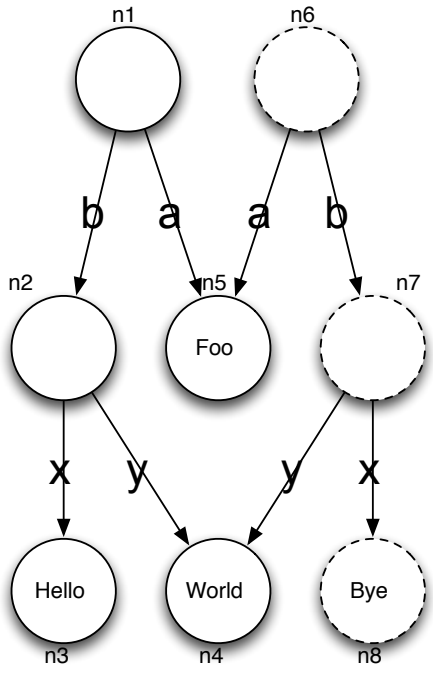


Figure 4.1: Immutable tree update

Note that in this setting, it is also trivial to handle symbolic links (to values or directories) as it is just a matter of using the right key.

4.1.6.4 History

The second high-level layer is a DAG data-structure to model the partial-order of hierarchy snapshots. A *snapshot* (or commit, or revision) is a node which contains some metadata, the key of the hierarchy root it is snapshoting and the list of keys of its immediate predecessors. This last part is encoding the Hasse relation of the partial order, which is equivalent to the full partial-order relation (as they have the same transitive closure). As the data-store is consistent, finding a common ancestor between two Irminsule instance is just a matter a finding a common snapshot key. An example of a DAG history is shown in Figure 4.2. The history is encoded as a DAG where full arrows depict the immediate predecessor relation and dotted arrows relate snapshots to hierarchy trees. In the example, c_1 is the first commit, with two branching futures: one is c_2, related to the hierarchy rooted at n_1 of Figure 4.1, where the hierarchy `b/x` contains `Hello`; the other future is c_3, related to the hierarchy rooted at n_6 where `b/x` contains `Bye`. c_4 is the result of a merge of c_1 and c_4 where we have (arbitrary) decided that `b/x` contains `Bye` (as n_6 is a child of c_4). We could have decided to pick `Hello` (and relate the commit to n_1) or any other contents (and create a new hierarchy root) instead.

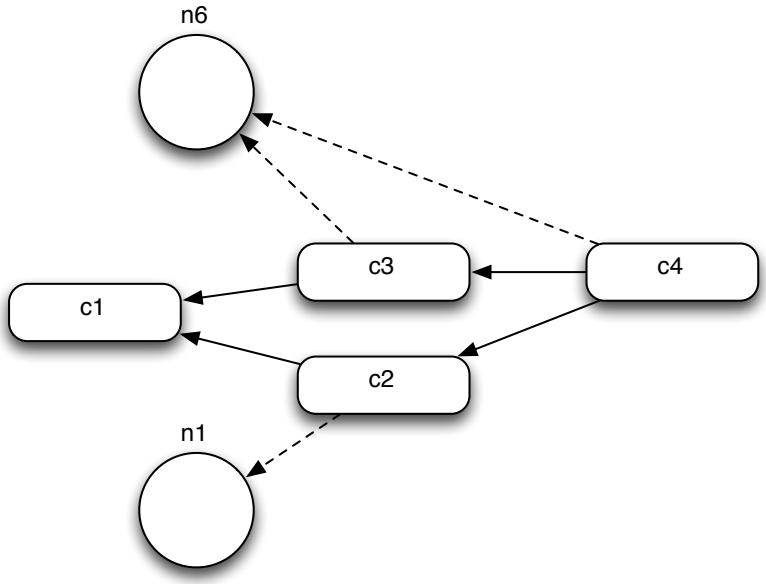


Figure 4.2: Partial-order of snapshots

4.1.6.5 Tag Data-store

Irminsule uses tags (or references) to pinpoint useful snapshots. The *tag data-store* is a key/value store, where keys are names created by users (and/or global names created by convention) and values are keys from the low-level data-store. The tag data-store is neither immutable nor consistent, so it is very different from the low-level one.

The tag store is central for higher-level algorithms such as:

Synchronization Different Irminsule instances can keep track of each-other by keeping a different tag for

each instance, pointing the associated state. These tags are updated each time an instance communicate with an other; this is very similar to the way vector clocks work (THOMAS: the original Git protocol does not synchronize tags between a group of instances, it's just peer-to-peer sync)

Garbage Collection As the low-level data-store is a growing-only database, it is crucial to regularly run an background process to clean-up the values which are not needed anymore. In order to do so, we need to know which snapshots are still useful and which are not.

4.2 Trevi

4.2.1 Introduction

Datacenters bring new challenges to the design and operation of storage systems. Several conflicting requirements need to be met at the same time, including scalability, data integrity [60, 8] and resilience, consistency and line-speed performance. Often, the only cost-effective solution is to relax some of the requirements. Over the years, client-server systems [106, 20, 40] have been succeeded by distributed systems that share functionality across multiple nodes in the network. In some cases, metadata servers are used to resolve the location of data and help maintaining an updated view of the storage resources so that consistency and data resilience is preserved in case of failures [25, 101, 126, 54]. Other systems distribute said functionality to multiple nodes or even across *all* storage nodes in order to support decentralisation and ease of management [146, 105, 94, 123, 120, 147].

Trevi runs on top of UDP or even raw Ethernet and does not depend on TCP. We leverage the inherent ability of fountain coding to multicast data blobs across storage nodes, and to utilise multiple replicas in parallel when reading data. Our approach is tolerant of losses and requires no retransmissions. Trevi requires no changes to the network stack; instead, we build on top of UDP or raw Ethernet frames, thus ensuring easier deployability.

Common to most existing systems is the necessity of balancing high throughput while keeping the deployment costs low. Consequently, they are built using commodity hardware and use TCP to communicate. The use of TCP leads to a number of known limitations that Trevi mitigates, as described next. This section is based on work presented at the Twelfth ACM Workshop on Hot Topics in Networks (HotNets-XII) in November 2013.

4.2.1.1 TCP Incast

A well known consequence of using TCP is TCP incast; *"a catastrophic TCP throughput collapse that occurs as the number of storage servers sending data to a client exceeds the ability of an Ethernet switch to buffer packets"* [108]. Incast is obvious for specific I/O workloads such as synchronised reads, but can also occur whenever severe congestion plagues the network, as TCP's retransmission timeouts are orders of magnitude higher than the actual Round Trip Times (RTTs) in datacenter networks. Several techniques have been proposed to mitigate TCP Incast [141, 158, 154], but their deployment is hindered due to requiring extensive changes in the OS kernel or the TCP protocol itself, or by requiring network switches to actively

monitor their queues and report congestion to end-nodes. Adopting fountain coding eliminates the need for retransmission upon packet loss. Instead, additional encoded symbols are transmitted until the receiver can decode the missing data (§4.2.3).

4.2.1.2 Trading network resources for resilience

Existing systems either send multiple copies of the same data [54, 146, 105] or apply an erasure code to the data and send the encoded pieces to multiple storage nodes [4, 35] to support resilience. The first approach effectively divides the performance of every write request by the number of stored replicas since each one of them has to be unicast separately. The latter does better in terms of required storage space, but erasure blocks need to be updated when writing to one or more blocks of data, and their old value must be fetched before the update. With fountain coding, write requests can be multicast to all replica points, thereby minimising network overhead and increasing energy efficiency (§4.2.4). Trevi, operating as a transport mechanism, is orthogonal to the usage of erasure codes as a means to provide storage resilience and it can work when either k-way replication or erasure coding is employed in the storage system.

4.2.1.3 Expensive switches to prevent packet loss

Realistic solutions to the TCP Incast problem are often solved by using network switches with large (and energy-hungry) memory for buffering packets in-flight. Packet loss or out-of-order receipt is much less important when using fountain coding. Consequently we can reduce the size of network buffers, or treat storage traffic as a lower QoS class with limited buffer space. Hence, our approach requires less energy to power the memory required for buffering storage requests (§4.2.3).

4.2.1.4 Lack of parallelism when multiple replicas exist

Systems that store copies of the same data in multiple locations (and where consistency among replicas is maintained or outdated replicas are flagged) only use a single storage node to fetch the data, leaving the rest of the nodes idle. Data reads are parallelised by striping a single blob to multiple disks and hoping that I/O requests are uniformly large. Usually, deep *read-ahead* policies are employed to force the system to fetch multiple stripes simultaneously, although this approach can be wasteful for workloads with small random reads. By contrast, fountain coding allows simultaneous multiple sources when reading data. This leads to more efficient utilization of storage resources even when the I/O workload cannot itself be parallelised in situations such as smaller read requests involving a single stripe (§4.2.5).

4.2.1.5 No (or basic) support for multipath transport

Most datacenters now offer multiple equal (or near-equal) cost paths through their fabric [56, 5] to support multipath transport, but exploiting these is hard. Protocol extensions like Multipath TCP [118] require extensive changes in the network stack. Other efforts seek to balance flows across different paths in a datacenter in a deterministic and rather static fashion [5, 68]. More dynamic approaches to balance packets across different paths to the same host are prohibitive because out-of-order packets can degrade TCP's performance significantly. Fountain coding schemes are much more welcoming to such dynamic balancing, since all sym-

bols are useful and there is no notion of out-of-order packets. Unlike TCP (where balancing happens on a per-flow basis to avoid out-of-order packets throughout a flow's lifetime), in our approach encoded symbols can be balanced independently. This provides a lot more flexibility in the design of in-network multipath mechanisms (§4.2.3).

4.2.2 A Strawman Design

We start with a strawman design that focuses on how blobs are transferred between clients and servers. We abstract out details of the OS integration (e.g. as a distributed block device, file system or key-value store), and omit details of how blobs are resolved to storage nodes, failure recovery and system expansion. Trevi can be integrated in any storage systems where blobs or pieces of blobs are assigned to storage servers deterministically.

Our description is based on a simplified version of the Flat Datacenter Storage (FDS) system [94]. In FDS, data is logically stored in blobs. A blob is a byte sequence named with a 128-bit GUID. The GUID can be selected by the application or assigned randomly by the system. Reads from and writes to a blob are done in units called tracts. Every disk is managed by a process called a *tract server* that serves read and write requests which arrive over the network from clients. FDS uses a metadata server, but its role during normal operations is simple and limited: collect a list of the system's active tract servers and distribute it to clients. This list is called the *tract locator table (TLT)*. In a single-replicated system, each TLT entry contains the address of a single tract server. With k-way replication, each entry has k tract servers. To read or write a tract from a blob, a client first selects an entry in the TLT by computing an index into it by hashing the blob's GUID and adding the index of the tract in the blob. This process is deterministic and returns the same set of tract servers to all clients that hold an up-to-date version of the TLT.

The only extension required to support Trevi is the addition of a column that stores some multicasting information (e.g. an IP multicast group) to the TLT. When nodes fail, or new nodes join the system, the TLT, which is cached locally to all clients, is updated [94]. Note that storage nodes subscribe to multicast groups in a deterministic fashion when they receive the TLT, and update their subscriptions when servers join or leave the storage network. Trevi's operation does not involve subscribing to and unsubscribing from multicast groups to transfer data among clients and servers.

4.2.2.1 Fountain coding-based blob transport

Fountain coding [82, 130] is central to our approach as it allows us to provide a storage service that is tolerant to packet loss, but without retransmissions or timeouts. It requires extra computing resources to encode and decode symbols and it has a small penalty in terms of bandwidth due to requiring a slightly larger number of encoded symbols than the initial number of fragments to decode the original information. This overhead can be as low as 5% [27] and proprietary raptor code implementations report a network overhead of less than 1%.[3]

[3] Source: Qualcomm website article "why raptor codes are better than reed-solomon codes for streaming applications"

As depicted in Figure 4.3, the sender calculates how many MTU sized fragments of the blob will be encoded in each symbol; this is called the *degree* of the encoded symbol. For example, the degree of encoded symbol *2* is 3. Selecting the degree is a crucial step in the process as it affects the efficiency of the information delivery in terms of the number of symbols required to decode the blob, and the decoding complexity.

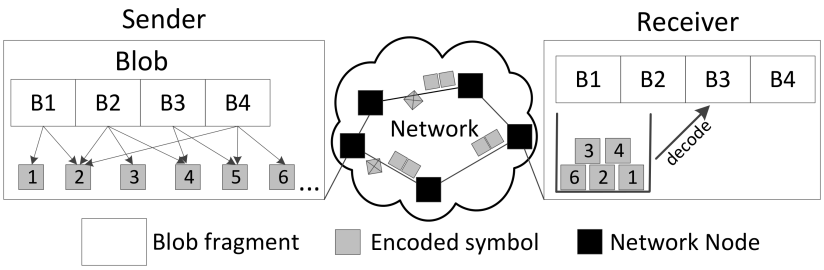


Figure 4.3: Fountain Coding Blobs

Different statistical distributions have been proposed for different coding techniques [82, 130]. The sender uniformly selects *degree* number of fragments and encodes them into the final symbol by XORing them; this set is called the symbol's *neighbours*. For instance, the neighbour set of symbol *2* includes fragments *B1*, *B2* and *B4*. A receiver utilizes symbols with degree 1 to partially or fully decode other symbols by XORing them with the decoded symbol. In Figure 4.3, the receiver utilizes the encoded symbol *1*, to decode fragment *B1*. It then XORs it with symbol *2*. *B2* is decoded using symbol *3*, which is also XORed with symbol *2*. Symbol *2*, then, only contains *B4*, which is decoded.

As shown in Figure 4.3, symbols may get lost on the way or travel via different paths. This is totally transparent to the endpoints. All encoded symbols contribute to the decoding process regardless the order they arrive, therefore there are no out-of-order symbols and, consequently, there is no need to keep or care about any sequencing information.

4.2.3 Receiver-driven flow control

Traditionally, the fountain coding transport model is *push* based. Senders start sending symbols and continue until all receivers have decoded the data and sent a notification to the sender or unsubscribed from a multicast group. In receiver-driven layered multicast [85], receivers play a more active role by subscribing to and unsubscribing from multicast groups that represent different coding layers, according to the network congestion.

In Trevi receivers actively manage the rate at which encoded symbols arrive (effectively providing flow and congestion control), by employing a receiver-driven *pull* communication scheme where a sender sends one or more symbols only when explicitly requested by a receiver. In §4.2.7.1, we discuss the applicability and benefits of a mixed, push/pull transport scheme.

Trevi receivers include a statistically unique label when requesting an encoded symbol so that the RTT can be calculated upon receiving a symbol sent in response to that request (and therefore carrying the same identifier). No action is taken when symbols are lost. We use labels instead of sequence numbers since

packets can arrive out-of-order.

A receiver-driven approach for requesting symbols simplifies flow and congestion control and guarantees that no extra symbols are sent after the receiver decodes the initial data. A receiver adjusts the number of pending symbol requests (called the *window* of requests) to handle changes in: 1) the rate at which a storage server can store data, 2) the rate at which a sender can send data and 3) the congestion in the network.

The first point has not been solved in past systems, especially for storage servers with spinning disks. In such cases the network bandwidth can be much higher than the disk array's throughput. Hence, there is no point in a storage server requesting more data than the amount it can actually store, sparing the extra bandwidth for others in the network. TCP's flow and congestion control would allow to transfer more data than what a server can actually store. This data would be buffered in memory until stored but some network bandwidth could have been spared to other data transfers.

The data rate of a sender is variable because it may serve multiple requests from receivers at the same time. Our approach ensures that senders will be requested to send encoded symbols at a rate that they can actually cope with. This rate can be achieved by adjusting the window of pending symbol requests when the RTT increases. Note that the RTT also increases when symbols are buffered in switches, but in both cases the window of pending requests should be decreased.

Congestion can be inferred and avoided by monitoring variations of RTTs for each symbol request. Receivers react when congestion occurs in the network by decreasing the number of pending symbols' requests. Additionally, losses in the network can be estimated since a receiver can know for which requests respective symbols did not arrive. It is worth highlighting that the notion of the window, as introduced above, is different from the classic TCP flow and congestion windows. There are no timeouts and no retransmissions in Trevi, and instead some internal timeouts which are only necessary to remove stale requests from the current window and update the loss statistics. These timers are adjusted based on the monitored RTTs but do not trigger any retransmission requests. In the worst case, if such a timer expires and an encoded symbol arrives after the respective request was removed, the receiver just increases the timer value for the upcoming requests; there is no penalty for the early timer expiration because the encoded symbol will be used in the decoding process just like any other symbol (there are no out-of-order symbols!).

We are still developing the details about the pace at which receivers populate the window of requests when starting to receive a new blob, as well as how they adjust its size when one of the three conditions mentioned above change and, therefore, we will not elaborate more on this subject, but we envision approaches similar to [19] and [134].

Packet losses are less important than in TCP; we do not identify packets with sequence numbers, we do not ask for specific packets and, consequently, there is no notion of retransmissions because of timeouts. Hence, there is no need to extensively buffer packets and desperately try to deliver them to their destination. Less buffering means cheaper and energy-efficient switches and/or more buffering for other TCP traffic which can

be clearly isolated from Trevi's traffic.

4.2.4 Multicasting data

Write requests for a (part of a) blob are first resolved utilising the hash of a blob's identifier to locate it in the Tract Locator Table (TLT) (step 1 in Figure 4.4). This gives the client the addresses of individual servers as well as the multicast group(s) to which it should send the tracts in the write request. Tract storage servers are assumed to be already subscribed to the right multicast groups, according to the information in the TLT (all entities in the storage network have the same view of the TLT [94]).

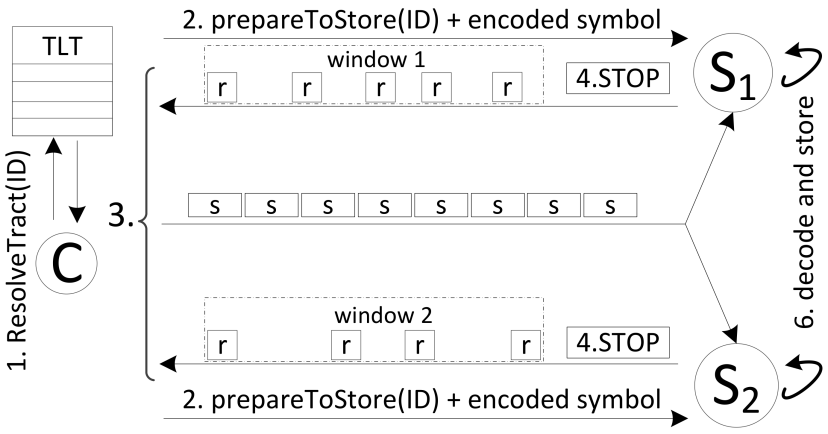


Figure 4.4: Multicasting Write Requests

When a client needs to write a tract to a number of servers, it first sends a *prepare* notification to all replica points (step 2). This notification must be sent in a reliable way, either via a separate control TCP connection or by employing a retransmission mechanism for this packet, which includes the identifier of the specific tract (an encoded symbol can be piggybacked in the notification). Upon receiving this notification, storage servers start requesting encoded symbols from the client (for write requests, servers are the receivers of symbols) (denoted as *r* in step 3).

It is important to note that although servers run their own flow and congestion control window, the client always sends encoded symbols based on the requests coming from the slowest server. The rest of the servers slow down the rate at which they request symbols to match the incoming rate (the separate windows of requests converge to the one of the slowest storage server). This way the client is able to multicast encoded symbols, denoted as *s* in step 3, to all servers at a rate defined by the slowest server (this feature is beneficial for the network because the multicasting rate is smoothed by the slowest server). Replication is by definition a synchronised operation which is completed only when all replicas acknowledge the reception of a tract. In §4.2.8, we describe a potential optimisation in case one storage server is straggling.

Finally, each server sends a *stop* notification containing the identifier of the stored tract, which must arrive to the client reliably (step 4). The client stops sending encoded symbols after receiving such notifications from all storage servers that store the specific tract.

In our approach data replication happens with the minimum network overhead and more energy-efficiency by

just multicasting data to a deterministically chosen set of nodes. Existing systems [146, 94, 105] select nodes for storing data deterministically, and therefore the only requirement is to have these nodes subscribing to a multicast group specific to the dataset assigned to them.

4.2.5 Multisourcing data

Reading tracts out of storage nodes follows similar lines. After resolving the nodes that store a specific tract (step 1 in Figure 4.5), a client sends a *get blob* request to all these nodes (step 2). All servers acknowledge the reception of the request (step 3) and a symbol can be piggybacked in the acknowledgement packet (Figure 4.5).

Figure 4.5: Multisourcing Read Requests

After receiving acknowledgements from all servers, the client simultaneously requests encoded symbols from all storage servers that hold an updated version of the tract (for read requests, the client is the receiver of symbols). Some systems [94, 146, 105] provide mechanisms to ensure that nodes with outdated data are never selected to fetch data. Trevi adopts a similar approach to ensure that such nodes will never be chosen. As shown in Figure 4.5, the client keeps separate windows of pending symbols' requests for each storage server.

Each storage node creates and sends encoded symbols in an independent and uncoordinated way in response to requests from the client (step 4). The only requirement here is that symbols created by storage nodes have a very high probability of being different, so we ensure storage nodes randomize the seed used when calculating the degree of each symbol. Different seeds will produce statistically different encoded symbols. Note that there is no need for any kind of synchronisation for this scheme to work. Servers transmit symbols at different rates (defined by the client's flow control mechanism) and each server only contributes the number of symbols that it is able to produce and transmit. Servers never send an encoded symbol unless they are requested to do so.

Finally, the client reliably sends a *stop* request (step 5) to (separately) let each server know that it decoded the requested blob. The whole procedure ends when the client passes the decoded blob to the application or the file or block subsystem (step 6).

The multisource transmission provided by Trevi allows storage resources to be fully utilized even when the I/O workload cannot be parallelised (e.g. for smaller read requests involving a single stripe). More specifically, all storage nodes that hold an updated version of some data can contribute to the transmission of the data to a client. This feature provides a second, inherent level of load balancing when fetching data, the first being the striping of blobs to multiple storage nodes.

4.2.6 The Price to Pay

In the previous sections we discussed what the problems are with existing storage systems that are based on commodity hardware and operate on top of TCP and how Trevi deals with them. In this section we discuss the potential downsides of our approach, which are all related with the fountain coding technique. We believe, though, that none of these issues is significant in a datacenter storage context.

CPU Overhead. Fountain coding involves encoding and decoding of information on the sender and receiver side, respectively. Here, the overhead comes from generating random numbers according to the used statistical distribution (e.g. the Robust Soliton Distribution [82]) and, mainly, from XORing several pieces of the initial information to produce each encoding symbol to be transmitted. We are confident that this overhead will not be prohibitive with respect to Trevi's applicability. First, modern hardware in datacenter networks consists of fast, multi-core CPUs that could easily cope with the encoding and decoding processes. Second, the process itself is highly parallelisable, thus one could take advantage of the multiple cores or even offload it to hardware (e.g. GPU or NetFPGA). Finally, an opportunistic approach, where a master replica decodes and stores the original blob while other servers serve the statistically-required number of symbols to decode the blob, can be used to minimize the overall CPU overhead of the storage system.

Network Overhead. As mentioned in §4.2.2.1, fountain coding involves a constant penalty in terms of network overhead. This overhead is not significant given that in Trevi, TCP incast, which can severely degrade the I/O performance, is mitigated, and that we save network resources by multicasting write requests.

Memory Overhead. In Trevi, a sender needs to have fast access to a blob of data as long as it creates new symbols (in response to respective requests). This implies that a blob must be in memory until all receivers successfully received and decoded the blob. This requirement could potentially have an impact on the required amount of memory to support multiple I/O requests in parallel. However, more control of the storage buffer cache (using direct I/O and a userspace cache, or even libOS techniques [83]) makes it possible to partially map larger blobs into memory to allow encoding to be suspended if the memory is required elsewhere. This helps to mitigate the tail of requests for a given blob, especially if there are stragglers in the storage cluster.

Energy Efficiency. We expect that Trevi will have a positive effect on energy consumption, although we will have to extensively evaluate this perspective with the system we are currently implementing. Trevi adds some energy intensive functionality; more processing power is required to encode and decode data, and slightly more data need to be transmitted compared to a regular TCP blob transfer. Additionally when multiple servers contribute to the transfer of data to a client multiple disks spin so that a number of encoded

symbols can be served. However, data can be multicast instead of multi-unicast and, because Trevi is tolerant to symbols' loss, buffers' size in network switches can be reduced.

4.2.7 Flow Control Refinements

The receiver-driven flow control of our strawman design can be refined in several ways.

4.2.7.1 Predictive Flow control

In order to minimize the overhead because of requesting encoded symbols separately, as described in §4.2.2, we could use a simple push-pull flow control scheme. When transmitting data, the source knows how many symbols approximately need to be transmitted in the absence of loss. This number depends on the statistical distribution that is used to calculate the degree of each symbol. Initially the source is set to send this much data and then pause. If no symbols have been lost, the source is notified by all receivers that the blob is decoded, and then it simply stops. In the opposite case, receivers start issuing pull requests for additional required symbols, as described in §4.2.2.

4.2.7.2 Priority and Scavenging

Fountain coding is inherently resilient to loss. This makes it suitable for scavenger-type QoS. In such systems, scavenger traffic receives a very low priority which means it will be preferentially dropped in the presence of any congestion. But in the absence of congestion, such scavenger traffic can be sent at near-line rate. In order to make best use of such a system the sender must rapidly detect how busy the network is. One way to do this would be to modify the measurement approach used for PCN.

Pre-Congestion Notification [87] is a measurement based admission control system for real time traffic. Traffic traversing each path through the network is viewed as a single combined flow, called an ingress egress aggregate. Central to PCN is the concept of a virtual queue – a simple mechanism that behaves like a queue with a slightly slower drain rate than the real queue.[4] As the virtual queue passes a lower threshold the queue is defined as being in a pre-congested state. If the virtual queue continues to grow it eventually passes a second threshold which indicates that the real queue is about to start to fill. In PCN, crossing either of these thresholds causes arriving packets to be marked, and the aggregate rate of marks is used to decide whether to admit new flows or to drop existing flows.

A similar virtual queue technique could be applied to our fountain storage system. This would allow the storage control nodes to assess how much other traffic is competing with the storage traffic. This can then be used to determine the safe rate at which to send data. This is particularly relevant for the case of multi-sourcing data from many replicas to one client machine. In this instance there is a very real risk of causing the final network queue nearest to the client to become congested. Simply running a virtual queue on this and using this as one of the parameters in the destination-driven flow control would significantly reduce this risk.

[4] Since 2010, all Broadcom router chipsets have natively supported a form of virtual queue called a threshold marker.

4.2.8 Optimising for slow writes

If one storage node is writing data much more slowly than the others in its group then it will have a disproportionate effect on the rate at which all others can write. There are two potential solutions to this problem. Firstly any node that is significantly slower could be removed from the multicast group. Secondly, the sender may choose to ignore the slow node and simply go faster than it can cope. If the node becomes overwhelmed it can simply unsubscribe from that multicast group and mark that tract as unreadable. Both of these approaches have implications for the degree of replication within the system, but may significantly improve performance.

4.2.9 Conclusions

Trevi overcomes the limitations present in all storage systems that are based on TCP. We described a strawman design which highlighted the main features of our approach, and also presented an initial design for a receiver-driven flow and congestion control mechanism that can better utilize storage and network resources in a datacenter storage network. Finally, we explored approaches for flow control refinements over our base mechanism.

We are implementing a complete storage system which incorporates Trevi based on the flat datacenter storage system [94] and built as a userspace application. We are also studying several strategies for adjusting the request window for the receivers according to changes in the storage workload and in the network, as described in §4.2.7. We will evaluate these mechanisms in multi-level datacenter topologies, and experiment with different flow control strategies using large scale simulations. Finally, we will explore the design space by using our mechanism in SDN-enabled topologies.

5 Prototypes, Testbed and Conclusions

5.1 Prototypes

The project has devoted a significant amount of effort to produce prototypes for the different mechanisms that we have designed. The implementations have been previously described in each of the sections where the different tools are presented. We include here a summary of the prototypes we have produced.

- Bandwidth liquidity
  - The Linux Kernel implementation of Multipath TCP, available at `http://multipath-tcp.org` is maintained as part of the Trilogy2 project. The maintenance of this implementation includes mostly bugfixing and porting the code to newer versions of the Linux Kernel. Among the newly introduced features are the support for hardware-offloading in MPTCP, support for the use of the Kernel's zero-copy feature and an improved support for the various kinds of middleboxes.
  - A proof of concept implementation of the new MPTCP uses cases proposed have been produced, as presented in Section 2
  - PVTCP prototype, ltproto, as described in Section 2
- CPU liquidity
  - Mobile virtual machine migration prototype
  - swBRAS prototype, as described in Section 3
- Storage liquidity
  - Irminsule - a file-system layer solution for distributing data, as described in Section 4
  - Trevi - fountain coding to tackle TCP issues of distributed storage, as described in Section 4

In addition to the prototypes of the different liquidity tools presented in this deliverable, we have also implemented prototypes of the different interaction of the different tools, presented in Deliverable 1.2, including GRIN and the use of MPTCP for preserving established connections in mobile devices.
The details of these implementations are contained in Deliverable 1.2.

5.2 Testbed

Trilogy2 seeks to expand on current state-of-the-art techniques and systems for creating and controlling the liquidity of resources. Trilogy2 continues with the work of MPTC and PVTCP to provide connections between resources. This works well at the protocol level for establishing reliable and deterministic communication channels. The overlaying management, the linked orchestration and the lifecycle management of resources is the next logical layer for controlling liquidity.

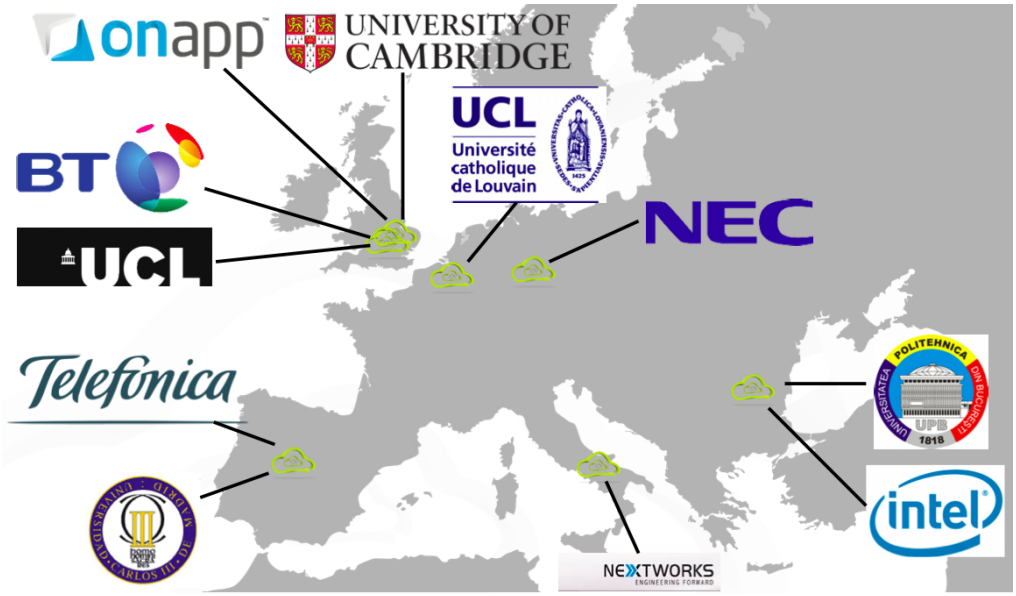


Figure 5.1: The envisaged Testbed platform

At a higher level the testbed platform looks to link geographically distributed resources that are exposed by means of the aforementioned tools. There are a number of technical challenges to meet distributing resources over the Internet as a whole. Exposure of resources requires a communication channel that all members that are part of the network can access and a description language or information model that can be used to describe resources. Work has been taken to detail the resources and protocols involved for resource communication in D2.2 - Information Model. The testbed platform is a shared resource that will be provided and shared by partners of the project. The geographic locations of the partners are shown in Figure 5.1. This effort ties in the resource exposure and offers insights into the management and control of resources across multiple geographical, ownership and network boundaries. Exposure and control of resources will naturally fit into the work of Tasks 2.3 and 2.4 (Incentives and Enforcement).

5.2.1 Partner Specific Motivations

Where applicable, the partners that would like to use the testbed for specific purposes have provided their motivations.

5.2.1.1 OnApp

OnApp would like to work on improving the interoperability of different cloud resources through working on and promoting a third party, open-source cloud brokerage system. By selecting and working on an independent solution, OnApp aim to push forward the idea and practical solution of allowing Cloud providers to pick and choose resources across a distributed resource pool. Market dynamics will come into play once the underlying technology has been created and the resources have been commoditised. There have been many proprietary systems or open-source systems that work with one or two commercial systems but OnApp believe that this model should be open and federated. Federated resource provision means that each cloud provider manages their own resources but also provides access and a set of rules that is established between

the different providers that allow resources to be used by a compliant agent. By working on the ideas behind the Testbed many different topics that are relevant to Trilogy2 will be covered. Enforcement and incentive mechanisms are areas that naturally evolve from market dynamics. The information model that enables for resources to be described is an evolving model that will fit in with the Trilogy2 architecture. Having a real system will allow some of the principles and ideas of the information model to be tested and implemented with real data sets being generated that can be used for analysis and evaluation. It is not the first testbed platform that has been attempted across a European setting but it is the first time that a commercial cloud platform provider has created such a platform to our knowledge.

5.2.1.2 Telefónica, I+D

The motivations of Telefónica, I+D for participating in the Trilogy2 testbed are to explore feasibility of the following key concepts:

- Stand-alone VM execution node only
  - Do we need to have the full OnApp infrastructure deployed?
  - Compare with other Open Source solutions
- Prioritised IPv6 access
  - What capabilities do we get when running in an environment where the IPv6 access if favoured over the IPv4 connectivity.

5.2.1.3 Managing Liquidity

OnApp as a partner of the consortium has a software platform that links Virtual Machine (VM) resources and can rapidly provision these resources based on a set of requirements provided to it. The platform serves as a good model for development of resource provisioning and also can act as a bootstrap infrastructure for having resources distributed geographically and under the control of different partners. The OnApp platform allows for customized VMs to be developed and 'templated'. Once ready, these templates can be provided to other partners and run as test VMs remotely. The testbed as such provides a rapid way for connecting the different partners' hardware infrastructure together and allow for test tools to be connected together. Most testing is normally carried out in the local infrastructure of the partner that is running the tests. By using a testbed, the type of testing can be increased and the breadth of tests expanded. OnApp are looking to use OCCI as a communication layer to the different resource providers. It is an open standard and as described in the standards section of the Information Model deliverable (D2.2) it has a sizeable following in the IaaS market. By using an open connection, Trilogy2 looks to break the vendor lock-in model that many other commercial cloud providers use at the moment. There are various implementations of OCCI connectors for different open-source projects and as part of the development work for exposing resources, OnApp has been working on an OCCI connector piece.

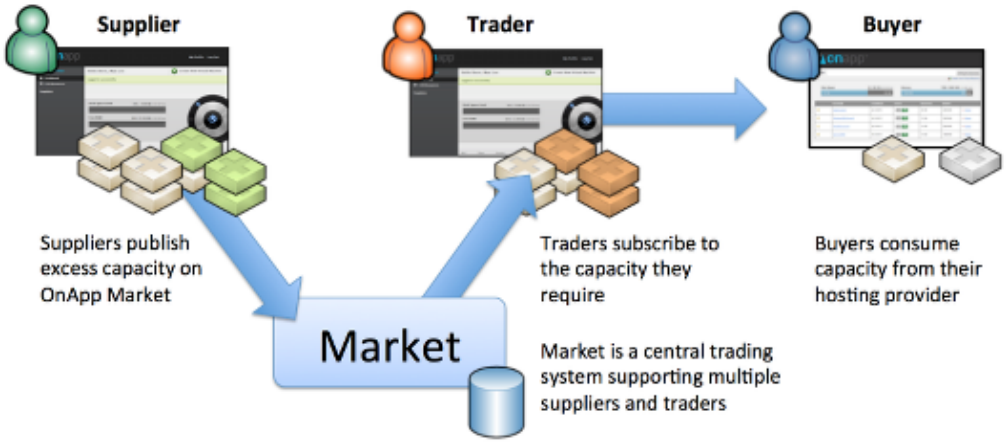


Figure 5.2: Roles within the market place

5.2.2 Infrastructure

To enable the test platform, OnApp have provided licenses to all of the T2 partners enabling them to create clouds that they can manage using a control panel that manages the resources. This allows a particular management system for creating and connecting Hypervisors, VMs, users and roles otherwise known as the dashboard. This is not the only system that will be supported by the testbed but the work for managing the testbed will be carried out on the OnApp platform as that is most familiar. A number of different cloud brokering services were investigated and for open source platforms there were not many available solutions. There are a number of commercial solutions that allow clouds to be provisioned and connected to each other. Most of the commercial solutions though support a number of public cloud providers. Some third party companies have tried to verify the reliability and robustness of public cloud providers by creating certifications such as that performed by RedHat [1].

Work has been carried out to create a market place that runs on the OnApp infrastructure that allows resources to be exposed and shared across a set of clouds.The market place has a number of roles for users that are associated with it as can be seen in Figure 5.2.

- The Supplier announces resources in the market place
- Trader is a role that acquires resources and then sells them
- A buyer ultimately consumes resources that provided by suppliers

5.2.2.1 Compatible One

CompatibleOne (`www.compatibleone.org`) is an opensource project that was created to provide a Cloud brokering service that was prototyped and developed as part of a French National Project. The project

[1] `http://www.redhat.com/solutions/cloud-computing/red-hat-cloud/find-public-cloud.html`

Company	Website
Amazon	http://aws.amazon.com
Google	https://cloud.google.com
HP	https://www.hpcloud.com
EMC	http://www.emc.com/cloud-virtualization/public-cloud.htm
Oracle	https://cloud.oracle.com
Rackspace	http://www.rackspace.com/cloud/
CenturyLink / Savvis	http://www.savvisdirect.com/cloud-servers
Salesforce	http://www.salesforce.com
Verizon / Terremark	http://www.verizonenterprise.com/cloud/
Joyent	http://joyent.com/products/compute-service
Citrix	http://www.citrix.com/cloudbridge
Bluelock	http://www.bluelock.com/cloud-services/
Microsoft	http://www.microsoft.com/en-us/server-cloud/public-cloud/
VMWare	http://vcloud.vmware.com/get_started

Table 5.1: Non-exhaustive list of public cloud and platform providers (Some information from [159])

Name	Website
CompatibleOne	http://www.compatibleone.org
CloudStack	http://cloudstack.apache.org
OpenStack	http://www.openstack.org
OpenNebula	http://nebula.nasa.gov
WS02	http://wso2.com/cloud/stratos/
EngineYard	https://www.engineyard.com/products/technology
OpsCode (/IBM) Chef	http://www.opscode.com/chef/#how-works
Opennodecloud	http://opennodecloud.com/about/

Table 5.2: Open source cloud-management systems

CO-PROCCI Name	Cloud Type	Cloud Website
AZPROCCI	Azure	http://www.windowsazure.com/en-us/
CNPROCCI	ComputeNext	https://www.computenext.com
DCPROCCI	DeltaCloud	http://deltacloud.apache.org
EZIPROCCI	EasiCloud	http://easi-clouds.eu
OAPROCCI	OnApp	http://www.onapp.com
ONPROCCI	Open Nebula	http://nebula.nasa.gov
OSPROCCI	OpenStack	http://www.openstack.org
PAPROCCI	ProActive	http://proactive.activeeon.com/index.php

Table 5.3: Types of PROCCIs available

was funded under FUI 10 [2]. The cloud brokering platform is called ACCORDS (Advanced Capabilities for CORDS) that relies on CORDS (CompatibleOne Resource Description System).
CompatibleOne aims to provide a vendor-neutral method for provisioning Cloud resources. The granularity of resource type that it can provision is a VM however the system is extensible and uses a REST interface for communicating between resource types and as such any resource that can present itself as a REST'ful service could theoretically be provisioned and managed from the platform.

5.2.2.2 ACCORDS

The platform has an internal representation of resources generated from descriptions it receives from Cloud providers. The interface between ACCORDS and a Cloud provider is known as a PROCCI. Using the OCCI interface for communication, ACCORDS will capture the information about the various operations such that it can then communicate with the resources (VMs) at a later point. It is the duty of the PROCCI developer to create the mappings between the ACCORDS platform and the Cloud provider platform. OnApp are creating a PROCCI interface as part of the work of T2 in order to manage resources. The management of the resources is also being investigated through changes in the Dashboard of OnApp. The understanding and techniques used to get to this point will be documented as part of the T2 work. As an Open-source platform, ACCORDS also has other PROCCI interfaces that already exist. A list of the PROCCI interfaces are included in Table 5.3.

5.2.3 Hardware Infrastructure

5.2.3.1 OnApp

Virtual CP server and multiple Hypervisors are available for testing various types of applications. Smallest disk drive size is 160GB and all the HVs are multi-core dual processors with 10GB+ RAM. All connected via IPv4. IP address for testbed will be in the form of *212.44.45.2xx*.

5.2.3.2 UC3M

CP: *163.117.140.121* and 1 * HV: *163.117.140.120*

[2] http://competitivite.gouv.fr/le-lancement-du-10e-appel-a-projets-de-r-d-du-fui/les-resultats-du-10e-appel-a-projets-701.html

5.2.3.3 NEC

CP: *195.37.154.9* and 1 * HV: *195.37.154.10*

5.2.3.4 Telefónica, I+D

The server is a quad-core processor Intel(R) Xeon(R) CPU X3363 @ 2.83GHz with 8Gbyte RAM and 500 Gbyte disk of disk storage. The IP connectivity is the following:

```
eth0 inet addr:10.95.12.244  Bcast:10.95.31.255  Mask:255.255.224.0
eth1 inet addr:192.168.0.79  Bcast:192.168.0.255  Mask:255.255.255.0
sit1 Link encap:IPv6-in-IPv4
inet6 addr: 2a02:9008:0:1902:8000::2/66 Scope:Global
```

The IPv6 address is public and directly connected to the IPv6 Internet via a corporate IPv6-in-IPv4 tunnel provider which is accessed through the corporate LAN, as shown in Figure 5.3.

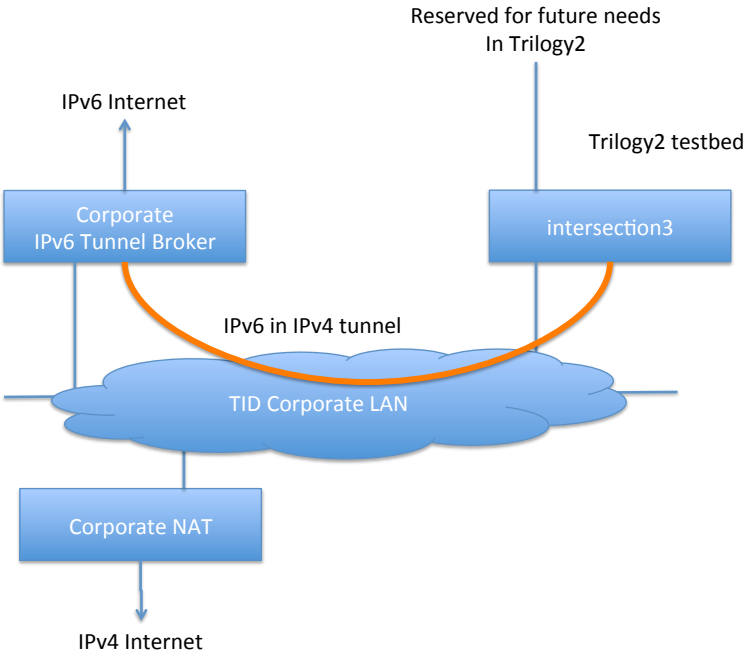


Figure 5.3: The TID testbed site

5.2.4 Authorisation / Use of Servers in Test-bed

There are different types of users: project-wide, partner-based, or person based. There is currently no distributed notion of a user, so all access credentials would need to be set up by each participating partner for all users that could access the system. We would recommend that the Control Panel server security is managed by a trusted and authorised person at each institution. For the Hypervisors it is likely that most partners will have one or two hypervisors that are set up for T2 work. Depending on how fluid we want to be, we could dynamically provision HVs using a system called CloudBoot, to allow us to choose the type of HV we need for particular tests. This may be important for some types of tests looking at low-level components of the

system and kernel level support. We expect though that most partners will set up an HV, and as long as it doesn't crash that it will just continue to run until the next test is scheduled. The HV type (Xen, Xen PV or KVM) should be recorded in the wiki page so we know the type of HV that we are running on. The discrete unit that we expect most partners will want to use is in terms of VMs. Partners will want to fire up VMs on different systems, run some performance benchmarks or some development work and then close down the connection. Possible issues include: the number of VMs that can be created, the maximum RAM/storage etc. that can be provisioned in one request, the total number of requests for resources in a given time, the network traffic, the number of VM restarts, the length of time a resource can be acquired and the types of services or applications that can be run.

Currently the part of the platform that allows the connectivity between partners is being worked on. However in the mean-time these issues listed above, among others, are likely to be the sort of things that partners may be concerned with. We would prefer not to be explicit with what is allowed, but rather disallow certain activities and try to envisage certain scenarios and pre-plan some work-around strategies. One possibility is to go down the route of creating a machine parseable *policy/robots.txt* file that resides on each partner's system. We would like the process to be automated, but there may need to be a human that signs off on resource usage, in which case contact details and procedure guidelines will be provided.

5.3 Concluding remarks

In this deliverable we described the progress on software platforms as the first year of the Trilogy2 project comes to an end. We presented solutions that enable resource pooling in each of the three main categories that the information model is built on: bandwidth, processing and storage. A significant amount of work has been done and is still ongoing based on Multipath TCP. For bandwidth, MPTCP is an important liqudity enabler, and we presented further developments related to its performance and security together with two novel use cases. Another promising avenue towards bandwidth liquidity is PVTCP, which plays a role complementary to MPTCP in settings where a sockets-based transport can be outperformed by employing specialized, more efficient mechanisms.

The section on processing liquidity presented two lines of work that follow increasingly important trends. The ubiquity of mobile devices can be readily observed in our everyday lives, and thus being able to seemingly migrate applications and data between them is an atractive proposition. Maybe less obvious, but no less important are the growing efforts to move away from doing a number of network processing taks in dedicated hardware boxes. The swBRAS is a step in this direction, as it allows efficient virtualization and offers increased flexibility. For storage liquidity we described two mechanisms that tackle the complexities found in a distributed setting. Irminsule aims to enable self-managing and self-healing systems while affording a great deal of portability. Trevi leverages the power of fountain codes to circumvent a lot of the shortcomings inherent to the use of TCP in most existing storage solutions. As previously mentioned, we have working implementations for most of the tools and platforms presented in this document. Where possible, we offered

different experimental results and dealt with important aspects such as potential issues related to performance or security. To enable further liquidity experiments, multiple partners are working to set up the Trilogy2 testbed. The focus of this document were tools for individual resource types. While it does make sense to create liquidity in this context, we also aim at cross-resource liquidity, as presented in Deliverable 1.2.

Bibliography

[1] 3GPP. UE "Fast Dormancy" behavior. Discussion and decision notes R2-075251, 2007.

[2] 3GPP. System impact of poor proprietary fast dormancy. Discussion and decision notes RP-090941, 2009.

[3] ABIResearch. Average size of mobile games for iOS increased by a whopping 42% between march and september. *http://www.webcitation.org/6F2Ifwv44*, 2012.

[4] M.K. Aguilera, R. Janakiraman, and Lihao Xu. Using erasure codes efficiently for storage in a distributed system. In *Proc. of DSN 2005*, 2005.

[5] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*, 2008.

[6] M. Allman. Comments on Bufferbloat. *ACM SIGCOMM Computer Communication Review*, 43(1), 2012.

[7] M. Allman and A. Falk. On the Effective Evaluation of TCP. *ACM SIGCOMM Computer Communication Review*, 29(5), 1999.

[8] Ross J. Anderson. The Eternity service. In *Pragocrypt*, 1996.

[9] O Andersson. *Experiment! Planning, Implementing and Interpreting*. Wiley, 2012.

[10] M. Bagnulo. Threat Analysis for TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6181 (Informational), March 2011.

[11] Marcelo Bagnulo, Christoph Paasch, Fernando Gont, Olivier Bonaventure, and Costin Raiciu. Analysis of MPTCP residual threats and possible fixes. Internet-Draft draft-bagnulo-mptcp-attacks-00, 2013.

[12] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proc. IMC*. ACM, 2009.

[13] Hitesh Ballani, Yatin Chawathe, Sylvia Ratnasamy, Timothy Roscoe, and Scott Shenker. Off by default. 2005.

[14] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 164–177, Bolton Landing, NY, USA, October 19–22 2003.

[15] S. Barre, C. Paasch, and O. Bonaventure. Multipath TCP: From Theory to Practice. In *IFIP Networking*, 2011.

[16] Martin Becke, Thomas Dreibholz, Hakim Adhari, and Erwin P. Rathgeb. On the fairness of transport protocols in a multi-path environment. pages 2666–2672, 2012.

[17] A. Bittau, D. Boneh, M. Hamburg, M. Handley, D. Mazieres, and Slack Q. Cryptographic protection of TCP Streams (tcpcrypt), 2012.

[18] G. Box and N. R. Draper. *Empirical Model Building and Response Surfaces*. John Wiley & Sons, 1987.

[19] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. TCP Vegas: new techniques for congestion detection and avoidance. In *SIGCOMM*, 1994.

[20] P. Breuer, A. Lopez, and A. Ares. The Network Block Device. *Linux Journal*, March 2000.

[21] Sebastian Burckhardt, Manuel Fähndrich, Daan Leijen, and Benjamin P. Wood. Cloud types for eventual consistency. In *Proceedings of the 26th European conference on Object-Oriented Programming*, ECOOP'12, pages 283–307, Berlin, Heidelberg, 2012. Springer-Verlag.

[22] Sebastian Burckhardt, Alexey Gotsman, and Hongseok Yang. Understanding eventual consistency, March 2013.

[23] Sebastian Burckhardt and Daan Leijen. Semantics of concurrent revisions. In Gilles Barthe, editor, *Programming Languages and Systems*, volume 6602 of *Lecture Notes in Computer Science*, pages 116–135. Springer Berlin Heidelberg, 2011.

[24] Sebastian Burckhardt, Daan Leijen, Manuel Fähndrich, and Mooly Sagiv. Eventually consistent transactions. In *Proceedings of the 21st European conference on Programming Languages and Systems*, ESOP'12, pages 67–86, Berlin, Heidelberg, 2012. Springer-Verlag.

[25] Philip H. Carns, Walter B. Ligon, III, Robert B. Ross, and Rajeev Thakur. PVFS: a parallel file system for Linux clusters. In *USENIX ALS*, 2000.

[26] Brian Carpenter.

[27] P. Cataldi, M.P. Shatarski, M. Grangetto, and E. Magli. Implementation and performance evaluation of LT and raptor codes for multimedia applications. In *IIH-MSP*, 2006.

[28] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain. TCP Fast Open. Internet Draft: draft-ietf-tcpm-fastopen-03, Work In Progress, 2013.

[29] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis. Increasing TCP's Initial Window. Internet Draft: draft-ietf-tcpm-initcwnd-08, Work In Progress, 2013.

[30] Cisco. Cisco Cloud Services Router 1000v Data Sheet. `http://www.cisco.com/en/US/prod/collateral/routers/ps12558/ps12559/data_sheet_c78-705395.html`, July 2012.

[31] Apache CloudStack. `http://cloudstack.apache.org/`.

[32] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM.

[33] Shuo Deng and Hari Balakrishnan. Traffic-aware techniques to reduce 3G/LTE wireless energy consumption. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, CoNEXT '12, pages 181–192, New York, NY, USA, 2012. ACM.

[34] Javier Díez, Marcelo Bagnulo, Francisco Valera, and Iván Vidal. Security for multipath tcp: A constructive approach. *Int. J. Internet Protoc. Technol.*, 6(3):146–155, November 2011.

[35] Alexandros G. Dimakis, Vinod Prabhakaran, and Kannan Ramchandran. Decentralized erasure codes for distributed networked storage. *IEEE Transactions on Information Theory*, 52:2809–2816, 2006.

[36] Mihai Dobrescu, Norbert Egi, Katerina Argyraki, Byung-Gon Chun, Kevin Fall, Gianluca Iannaccone, Allan Knies, Maziar Manesh, and Sylvia Ratnasamy. Routebricks: exploiting parallelism to scale software routers. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 15–28, New York, NY, USA, 2009. ACM.

[37] Philip Eardley. Survey of MPTCP Implementations. Internet-Draft draft-eardley-mptcp-implementations-survey-02, 2013.

[38] Norbert Egi, Adam Greenhalgh, Mark Handley, Mickael Hoerdt, Felipe Huici, and Laurent Mathy. Towards high performance virtual routers on commodity hardware. In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, pages 20:1–20:12, New York, NY, USA, 2008. ACM.

[39] Julian Elischer and Archie Cobbs.

[40] Lans Ellenberg. DRBD 9 and device-mapper: Linux block level storage replication. In *the Linux System Technology Conference*, 2009.

[41] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.

[42] Dino Farinacci, V. Fuller, D. Meyer, and D. Lewis. Rfc 6830: The locator/id separation protocol (lisp), January 2013.

[43] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827 (Best Current Practice), May 2000. Updated by RFC 3704.

[44] R. A. Fisher et al. *The Design of Experiments*. Number 5th ed. Oliver and Boyd, London and Edinburgh, 1949.

[45] Project Floodlight. `http://www.projectfloodlight.org/floodlight/`.

[46] Florin Sultan and Kiran Srinivasan and Deepa Iyer and Liviu Iftode. Migratory TCP: Highly Available Internet Services Using Connection Migration. Rutgers University Technical Report DCS-TR-462.

[47] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance, 1993.

[48] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar. Architectural Guidelines for Multipath TCP Development. RFC6182, March 2011.

[49] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC6824, January 2013.

[50] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. Rfc6824, IETF, 2013.

[51] Andres Garcia-Saavedra, Pablo Serrano, Albert Banchs, and Giuseppe Bianchi. Energy consumption anatomy of 802.11 devices and its implication on modeling and design. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, CoNEXT '12, pages 169–180, New York, NY, USA, 2012. ACM.

[52] Aaron Gember, Ashok Anand, and Aditya Akella. A comparative study of handheld and non-handheld traffic in campus wi-fi networks. In *Proceedings of the 12th international conference on Passive and active measurement*, PAM'11, pages 173–183, Berlin, Heidelberg, 2011. Springer-Verlag.

[53] J. Gettys and K. Nichols. Bufferbloat: Dark buffers in the internet. *ACM Queue*, 9(11), 2011.

[54] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *SOSP*, 2003.

[55] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, June 2002.

[56] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. *ACM SIGCOMM CCR*, 39(4), 2009.

[57] Michael B. Greenwald, Sanjeev Khanna, Keshav Kunal, Benjamin C. Pierce, and Alan Schmitt. Agreeing to agree: Conflict resolution for optimistically replicated data. In Shlomi Dolev, editor, *International Symposium on Distributed Computing (DISC)*, 2006.

[58] Saikat Guha and Paul Francis. An end-middle-end approach to connection establishment. In *SIGCOMM*, SIGCOMM '07, pages 193–204, New York, NY, USA, 2007. ACM.

[59] Sangjin Han, Scott Marshall, Byung-Gon Chun, and Sylvia Ratnasamy. Megapipe: a new programming interface for scalable network i/o. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, OSDI'12, pages 135–148, Berkeley, CA, USA, 2012. USENIX Association.

[60] Steven Hand and Timothy Roscoe. Mnemosyne: Peer-to-Peer steganographic storage. In *IPTPS*, 2002.

[61] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible Network Experiments using Container-based Emulation. In *ACM CoNext*, 2012.

[62] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC6582, April 2012.

[63] Takahiro Hirofuchi, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi. Enabling instantaneous relocation of virtual machines with a lightweight vmm extension. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 73–83, Washington, DC, USA, 2010. IEEE Computer Society.

[64] Takahiro Hirofuchi, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi. Reactive consolidation of virtual machines enabled by postcopy live migration. In *Proceedings of the 5th International Workshop on Virtualization Technologies in Distributed Computing*, VTDC '11, pages 11–18, New York, NY, USA, 2011. ACM.

[65] Takahiro Hirofuchi, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi. Reactive cloud: Consolidating virtual machines with postcopy live migration. *IPSJ Transactions on Advanced Computing Systems*, pages 86–98, 2012.

[66] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. Is it still possible to extend tcp? In *Proc. ACM IMC*, 2011.

[67] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. Is it still possible to extend tcp? In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pages 181–194, New York, NY, USA, 2011. ACM.

[68] C. Hopps. Analysis of an equal-cost multi-path algorithm. *RFC 2992*, 2000.

[69] Junxian Huang, Feng Qian, Alexandre Gerber, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. A close examination of performance and power characteristics of 4g lte networks. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, MobiSys '12, pages 225–238, New York, NY, USA, 2012. ACM.

[70] Pan Hui, Richard Mortier, Kuang Xu, Jon Crowcroft, and Victor O. K. Li. Sharing airtime with shair avoids wasting time and money. In *Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, HotMobile '09, pages 6:1–6:6, New York, NY, USA, 2009. ACM.

[71] iMatrix Corps.

[72] European Telecommunications Standards Institute. Industry specification group.

[73] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. RFC 3775 (Proposed Standard), June 2004. Obsoleted by RFC 6275.

[74] D. Kaspar. *Multipath Aggregation of Heterogeneous Access Networks*. PhD thesis, University of Oslo, 2011.

[75] Ethan Katz-Bassett, David R Choffnes, Ítalo Cunha, Colin Scott, Thomas Anderson, and Arvind Krishnamurthy. Machiavellian routing: improving internet availability with bgp poisoning. In *Hotnets*, page 11. ACM, 2011.

[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 *ACM CoNext*, 2012.

[77] S. Kleijnen, J.and Sanchez, T. Lucas, and T. Cioppa. State-of-the-art Review: a User's Guide to the Brave new World of Designing Simulation Experiments. *INFORMS Journal on Computing*, 17(3), 2005.

[78] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, August 2000.

[79] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, August 2000.

[80] Cheng Li, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno Preguiça, and Rodrigo Rodrigues. Making geo-replicated systems fast as possible, consistent when necessary. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, OSDI'12, pages 265–278, Berkeley, CA, USA, 2012. USENIX Association.

[81] Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. Don't settle for eventual: scalable causal consistency for wide-area storage with cops. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 401–416, New York, NY, USA, 2011. ACM.

[82] Michael Luby. LT Codes. In *Proc. of FOCS*, 2002.

[83] Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. Unikernels: library operating systems for the cloud. In *ASPLOS*, 2013.

[84] Joao Martins, Mohamed Ahmed, Costin Raiciu, and Felipe Huici. Enabling fast, dynamic network processing with clickos. *To appear in SIGCOMM HotSDN*, 2013.

[85] Steven McCanne, Van Jacobson, and Martin Vetterli. Receiver-driven layered multicast. In *SIGCOMM*, 1996.

[86] Liam McNamara, Cecilia Mascolo, and Licia Capra. Media sharing based on colocation prediction in urban transport. In *Proceedings of the 14th ACM international conference on Mobile computing and networking*, pages 58–69, 2008.

[87] Michael Menth, Frank Lehrieder, Bob Briscoe, Philip Eardley, Toby Moncaster, et al. A survey of PCN-based admission control and flow termination. *Communications Surveys & Tutorials, IEEE*, 12(3):357–375, 2010.

[88] M. Morris and T. Mitchell. Exploratory Designs for Computational Experiments. *Journal of Statistical Planning and Inference*, 43(3), 1995.

[89] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson. Rfc 5201: Host identity protocol, April 2008.

[90] R. Myers and D. Montgomery. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. Wiley, 2009.

[91] British Telecom FON Network. http://www.btfon.com/.

[92] H. X. Nguyen and M. Roughan. Rigorous Statistical Analysis of Internet Loss Measurements. *IEEE/ACM Transactions on Networking*, 38(1), 2012.

[93] Kathleen Nichols and Van Jacobson. Controlling queue delay. *Queue*, 10(5):20:20–20:34, May 2012.

[94] Edmund B. Nightingale, Jeremy Elson, Jinliang Fan, Owen Hofmann, Jon Howell, and Yutaka Suzue. Flat datacenter storage. In *USENIX OSDI*, 2012.

[95] E. Nordmark and M. Bagnulo. Rfc 5533: Shim6: Level 3 multihoming shim protocol for ipv6, June 2009.

[96] E. Nordmark and M. Bagnulo. Shim6: Level 3 Multihoming Shim Protocol for IPv6. RFC 5533 (Proposed Standard), June 2009.

[97] Erik Nordström, David Shue, Prem Gopalan, Robert Kiefer, Matvey Arye, Steven Y. Ko, Jennifer Rexford, and Michael J. Freedman. Serval: an end-host stack for service-centric networking. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, NSDI'12, pages 7–7, Berkeley, CA, USA, 2012. USENIX Association.

[98] OpenDaylight Project. ,`http://www.opendaylight.org/`.

[99] OpenFlow Switch Specification. `https://www.opennetworking.org/sdn-resources/onf-specifications`.

[100] OpenStack Cloud software. `http://www.openstack.org/`.

[101] Oracle. The Oracle Clustered File System. http://oss.oracle.com/projects/ocfs/.

[102] C. Paasch and O. Bonaventure. Securing the MultiPath TCP handshake with external keys, October 2012.

[103] C. Paasch, G. Detal, F. Duchene, C. Raiciu, and O. Bonaventure. Exploring Mobile/WiFi Handover with Multipath TCP. In *ACM SIGCOMM workshop CellNet*, 2012.

[104] Christoph Paasch and Olivier Bonaventure. Securing the MultiPath TCP handshake with external keys. Internet-Draft draft-paasch-mptcp-ssl-00, 2012.

[105] George Parisis, George Xylomenos, and Theodore Apostolopoulos. DHTbd: A reliable block-based storage system for high performance clusters. In *CCGRID*, 2011.

[106] Brian Pawlowski, David Noveck, David Robinson, and Robert Thurlow. The NFS version 4 protocol. In *SANE 2000*, 2000.

[107] Colin Perkins. RFC 2002: IP Mobility Support, October 1996.

[108] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Srinivasan Seshan. Measurement and analysis of TCP throughput collapse in cluster-based storage systems. In *USENIX FAST*, 2008.

[109] Benjamin C. Pierce and Jerome Vouillon. Unison: A file synchronizer and its specification. In Naoki Kobayashi and Benjamin C. Pierce, editors, *TACS*, volume 2215 of *Lecture Notes in Computer Science*, page 560. Springer, 2001.

[110] Lucian Popa, Costin Raiciu, Ion Stoica, and David Rosenblum. Reducing congestion effects in wireless networks by multipath routing. In *Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols*, ICNP '06, pages 96–105, Washington, DC, USA, 2006. IEEE Computer Society.

[111] Feng Qian, Zhaoguang Wang, Alexandre Gerber, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. Top: Tail optimization protocol for cellular radio resource allocation. In *Proceedings of the The 18th IEEE International Conference on Network Protocols*, ICNP '10, pages 285–294, Washington, DC, USA, 2010. IEEE Computer Society.

[112] Feng Qian, Zhaoguang Wang, Alexandre Gerber, Zhuoqing Morley Mao, Subhabrata Sen, and Oliver Spatscheck. Characterizing radio resource allocation for 3g networks. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, IMC '10, pages 137–150, New York, NY, USA, 2010. ACM.

[113] Quagga Routing Suite. `http://www.nongnu.org/quagga/`.

[114] C. Raiciu, M. Handley, and D. Wischik. Coupled Congestion Control for Multipath Transport Protocols. RFC6356, October 2011.

[115] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley. How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP. In *USENIX NSDI*, 2012.

[116] Costin Raiciu, Dragoș Niculescu, Marcelo Bagnulo, and Mark James Handley. Opportunistic mobility with multipath tcp. In *Proceedings of the sixth international workshop on MobiArch*, MobiArch '11, pages 7–12, New York, NY, USA, 2011. ACM.

[117] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How hard can it be?... In *NSDI*, NSDI'12, pages 29–29, Berkeley, CA, USA, 2012. USENIX Association.

[118] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How hard can it be? designing and implementing a deployable multipath TCP. In *Proc. of USENIX NSDI*, 2012.

[119] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and Analysis of Computer Experiments. *Statistical science*, 4(4), 1989.

[120] Yasushi Saito, Svend Frolund, Alistair C. Veitch, Arif Merchant, and Susan Spence. FAB: building distributed enterprise disk arrays from commodity components. In *ASPLOS*, 2004.

[121] Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, March 2005.

[122] J. Santiago, M. Claeys-Bruno, and M. Sergent. Construction of Space-Filling Designs using WSP Algorithm for High Dimensional Spaces. *Chemometrics and Intelligent Laboratory Systems*, 113, 2012.

[123] Frank Schmuck and Roger Haskin. GPFS: A shared-disk file system for large computing clusters. In *of USENIX FAST*, 2002.

[124] Simon Schuetz, Nikolaos Koutsianas, Lars Eggert, Wes Eddy, Yogesh Swami, and Khiem Le. TCP Response to Lower-Layer Connectivity-Change Indications. Internet-Draft draft-schuetz-tcpm-tcp-rlci-03, 2008.

[125] Aaron Schulman, Vishnu Navda, Ramachandran Ramjee, Neil Spring, Pralhad Deshpande, Calvin Grunewald, Kamal Jain, and Venkata N. Padmanabhan. Bartendr: a practical approach to energy-aware cellular data scheduling. In *Proc. Mobicom*. ACM, 2010.

[126] Philip Schwan. Lustre: Building a file system for 1,000-node clusters. In *Linux Symposium*, 2003.

[127] J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP Buffer Tuning. *ACM SIGCOMM Computer Communication Review*, 28(4), 1998.

[128] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *13th Int. Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS)*, Grenoble, France, October 2011. Springer.

[129] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: network processing as a cloud service. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 13–24, New York, NY, USA, 2012. ACM.

[130] Amin Shokrollahi. Raptor codes. *IEEE Transactions on Information Theory*, 52(6):2551–2567, 2006.

[131] Alex C. Snoeren and Hari Balakrishnan. An end-to-end approach to host mobility. In *Mobicom*, MobiCom '00, pages 155–166, New York, NY, USA, 2000. ACM.

[132] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), September 2007. Updated by RFCs 6096, 6335, 7053.

[133] Florin Sultan, Aniruddha Bohra, and Liviu Iftode. Service continuations: An operating system mechanism for dynamic migration of internet service sessions. In *SRDS*. IEEE Computer Society, 2003.

[134] Kun Tan and Jingmin Song. A Compound TCP approach for high-speed and long distance networks. In *IEEE INFOCOM*, 2006.

[135] D. B. Terry, M. M. Theimer, Karin Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *Proceedings of the fifteenth ACM symposium on Operating systems principles*, SOSP '95, pages 172–182, New York, NY, USA, 1995. ACM.

[136] AT&T, BT, CenturyLink, China Mobile, Colt, Deutusche Telekom, KDDI, NTT, Orange, Telefom Italia, Telefonica, Telstra, and Verizon. Network function virtualization - white paper.

[137] Broadband forum. Tr101v2 migration to ethernet-based dsl aggregation - issue 2, 2011.

[138] OECD. Broadband portal http://dx.doi.org/10.1787/888932398138.

[139] Javier Ubillos and Zhongxing Ming.

[140] Narseo Vallina-Rodriguez, Vijay Erramilli, Yan Grunenberger, Laszlo Gyarmati, Nikolaos Laoutaris, Rade Stanojevic, and Konstantina Papagiannaki. When david helps goliath: the case for 3g onloading. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 85–90, New York, NY, USA, 2012. ACM.

[141] Vijay Vasudevan, Amar Phanishayee, Hiral Shah, Elie Krevat, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Brian Mueller. Safe and effective fine-grained TCP retransmissions for datacenter communication. In *SIGCOMM*, 2009.

[142] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *J. Network Syst. Manage.*, 13(2):197–217, 2005.

[143] Open vSwitch. Production quality, multilayer open virtual switch.

[144] Vyatta. The Open Source Networking Community. http://www.vyatta.org/, July 2012.

[145] Michael Walfish, Jeremy Stribling, Maxwell Krohn, Hari Balakrishnan, Robert Morris, and Scott Shenker. Middleboxes no longer considered harmful. In *OSDI*, OSDI'04, pages 15–15, Berkeley, CA, USA, 2004. USENIX Association.

[146] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: a scalable, high-performance distributed file system. In *USENIX SOSP*, 2006.

[147] Brent Welch, Marc Unangst, Zainul Abbasi, Garth Gibson, Brian Mueller, Jason Small, Jim Zelenka, and Bin Zhou. Scalable performance of the Panasas parallel file system. In *USENIX FAST*, 2008.

[148] Michael Welzl and Stefan Jorer. Towards a protocol-independent internet transport api. 2012.

[149] Open Networking Foundation white paper. Software-Defined Networking: The New Norm for Networks, April 2012.

[150] John Wiegley. Git from the bottom up, May 2008.

[151] D. Wing and A. Yourtchenko. RFC 6555: Happy Eyeballs: Success with Dual-Stack Hosts, April 2012.

[152] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. In *USENIX NSDI*, 2011.

[153] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, implementation and evaluation of congestion control for multipath tcp. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11, pages 8–8, Berkeley, CA, USA, 2011. USENIX Association.

[154] Haitao Wu, Zhenqian Feng, Chuanxiong Guo, and Yongguang Zhang. ICTCP: Incast congestion control for TCP in data center networks. In *Proceedings of CoNEXT*, 2010.

[155] J. Wu, J. Bi, M. Bagnulo, F. Baker, and C. Vogt. Source Address Validation Improvement (SAVI) Framework. RFC 7039 (Informational), October 2013.

[156] Fengyuan Xu, Yunxin Liu, Thomas Moscibroda, Ranveer Chandra, Long Jin, Yongguang Zhang, and Qun Li. Optimizing background email sync on smartphones. 2013.

[157] B. Zhang, T. Ng, A. Nandi, R. Riedi, P. Druschel, and G. Wang. Measurement-Based Analysis, Modeling, and Synthesis of the Internet Delay Space. *IEEE/ACM Transactions on Networking*, 18(1), 2010.

[158] Yan Zhang and N. Ansari. On mitigating TCP incast in data center networks. In *Proc. of IEEE INFOCOM*, 2011.

[159] Top 10 cloud computing providers of 2012. `http://searchcloudcomputing.techtarget.com/photostory/2240149039/Top-10-cloud-providers-of-2012/2/10-VMware#contentCompress`.

[160] Yocto Project. `https://www.yoctoproject.org/`.

