# Serial Experiments Online

Juan J. Ramos-Munoz
Signal Theory, Telematics and Communications Department
University of Granada
Granada, Spain
jjramos@ugr.es

Lidia Yamamoto
Computer Networks Group
Computer Science Department
Bernoullistrasse 16, CH-4056
Basel, Switzerland
lidia.yamamoto@unibas.ch

Christian Tschudin
Computer Networks Group
Computer Science Department
Bernoullistrasse 16, CH-4056
Basel, Switzerland
christian.tschudin@unibas.ch

## ABSTRACT

Current network protocols must comply with rigid interfaces and rules of behavior to fit into well defined, vertical protocol stacks. It is difficult for network designers to offer a wide spectrum of alternative protocols suitable for diverse situations, and to make the stack evolve to match new needs. The tendency is to design protocols that can adapt to the widest possible spread of use. However, even the best adaptive protocols cannot possibly cope with all situations. When their adaptivity limits are reached, the ability to switch to other protocols becomes a clear advantage.

Our aim in this paper is to present *Lightweight Autonomous resIlient Networks* (LAIN), a framework that exploits the multiplicity of alternative protocol, and exposes the spectrum of choice to the advantage of the applications. The system runs continuous experiments with alternative protocols online, in parallel as well as serially, in order to select automatically those that best match the application's needs under the current network conditions. We report first results on the feasibility of the approach and point out the need for such a system in network and protocol evolution.

## Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Reliability, availability, and serviceability; C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network monitoring*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design Network communications—*Network communications*

## General Terms

Design, Performance, Reliability

## Keywords

Autonomic network, self-evaluation, network architecture, network evolution, runtime protocol switching, knowledge plane.

## 1. INTRODUCTION

Currently, a protocol tends to be considered as more evolved than its predecessor if it is applicable in more contexts and situations. In our opinion, this leads to a wrong filter in protocol and service design and suboptimal network operations because it establishes an artificially high barrier against better, although perhaps specialized, solutions. It also creates a barrier against innovation and imposes the need to decide whether to design for the worst case or for the average case [7]. In both cases many situations will not be suitably covered, and many useful protocols will not be exploited.

We envisage a networking environment where myriads of protocols may coexist and compete with each other to provide the best service. Minor, restricted or even partially defective protocols may be exploited on a regular basis, provided that they are useful in some context. In this paper we present LAIN, an experimental framework to explore the idea of online dynamic protocol switching. The LAIN framework permits the coexistence of alternative protocol instances that run in parallel or serially, and are automatically selected at run time.

Resilience to protocol failures or misbehavior is of paramount importance in this context, as a full model comprising the behaviors of all possible protocols would be unrealistic and cannot be counted upon. The framework is thus based on the premise of doing continuous experiments on running protocols, by constantly monitoring their behavior, measuring their performance, and selecting the most suitable protocols accordingly.

### 1.1 Online Protocol Choice

The approach of *protocol, service and server switching* is not new and is increasingly applied in the Internet. Starting from low in the stack, we point out fall-back solutions in modems depending on the peer's capability and line conditions. At the network level, Skype [1], in the same way as firewall tunneling protocols [21], try out several venues until connectivity, as well as performance, matches the required thresholds. Regarding transport, on-the-fly compression can be enabled depending on whether the TCP queue becomes sufficiently filled [14]. Many fault tolerance mechanisms rely on alternative instances they can easily switch among, like DNS secondary name servers or the micro-rebooting of servers [2]. Protocol switching can even be applied to full protocol stacks as is the case for handover in wireless networking and software defined radios, where either the parallel operation of different stacks, or a complete switch of stacks is part of normal operation. So far, all these "switching" activities were seen as context-specific solutions.

We propose to raise this scattered switching logic to the level of a knowledge plane which can make informed decisions within a well established framework and with sound, reusable techniques. For that to happen in a robust way, the system needs to monitor constantly the performance of each protocol and compare it with a desired goal. Alterna-

tive protocols implementing the same functionality in different ways compete with each other: The system selects the most suitable one for each situation. This selective pressure combined with self-inspection results in a system that always seeks to deliver the expected service by moving to a better protocol when the previous one does not comply to the expectations.

## 1.2 Experiments and Switching Logic

One possibility to select protocols automatically in a dynamic way is to run a productive or current best practice version in parallel with an experimental, potentially better version and continuously compare their performance. The productive version actually delivers the service, and switching to the experimental one only occurs when the latter becomes clearly superior to the former. This approach obviously consumes extra resources, since parallel, alternative trial channels are maintained; but it could be regarded as a price to pay for robustness.

However, doubling the resources is just a first approach to the problem. Better allocation schemes, in which the experimental channel is also used for (partial) service delivery, can be envisaged. How exactly to allocate resources, how to assess the matching of expectations, when (and when not) to switch etc. – all these questions are discussed in this paper, and positioned in a framework for future network research and development.

The contribution of this paper is threefold: First, we describe the elements of a self-selective network in Sect. 2. Second, we show the operation and benefits of case-by-case decisions applied to two different contexts, one at transport level and another in a voice over IP system by means of simulations in Sects. 3 and 4 respectively. Finally, in Sect 5 we discuss the selection approach compared to existing systems and in the light of future networks supporting automatic protocol evolution.

# 2. ONLINE PROTOCOL EXPERIMENTS

In an ideal networking world, choosing among two potential services could be based on full knowledge of the factors that determine the services' fitness for a given task and situation. This knowledge can be gained through analytic work and/or through past experiences. Our hypothesis is that a complete modeling will less and less be possible, that network state can not always be sufficiently gathered, and that not all implementations will adhere to the supposed model anyway. Online protocol experiments are a necessary addition to existing assessment methods and can even be used in cases where no model is available. In this section, we structure the problem of automatic protocol switching and elaborate on the problems of doing continuous evaluations, both of productive as well as experimental services.

## 2.1 Framework

Fig. 1 shows the elements needed for implementing a framework for online service evaluation and selection. A sample application is displayed on top. Every application using the framework has its own instance of the protocol selection framework. Therefore each instance can be adapted to the specific requirements of an application.

The elements of the framework are:

- *Protocol functions:* Pool of protocol candidates from which our system can choose. Depending on the granularity we are looking at, these are complete stacks, single protocols, sub-protocol functions like stream splitters, as well as access to remote service instances.

  In order to perform the components selection, the protocol functions are grouped into services. A service will be defined as a well-known set of tasks, with a well-known interface. Each alternative protocol implementation must adhere to the proper service specification, in order to be eligible as a candidate in the test round. Service profiles include an input, output, and monitoring interface. While input and output interfaces rule the interaction with the application and the underlaying services, monitoring interfaces export performance parameters (such as correct packets received) to the service checker. This profile also states the functional requirements for the service (it is to say, which support functions are needed). For instance, a reliable transport service may require a system function to calculate an error correction code.

- *Monitor helpers:* Measure parameters from the network and from running protocol instances and provide this information to the experimenter. The monitors helpers are dynamic entities that can be activated or deactivated on demand. Moreover, new monitors can be added to the framework, as long as they provide a proper interface description.

- *Service checker:* Verifies whether a given service complies to the application request, signaling any deviations to the experimenter. To this end, it uses a fitness function which measures the performance of each protocol function. It may be a combination of objective (goals and observable performance parameters) and subjective (perceived quality of service) components. Since the criteria of the service evaluation are determined by the particular requesting application and user's preferences, the fitness function should be provided by the service petitioner. The fitness function can use any monitored parameter to perform its

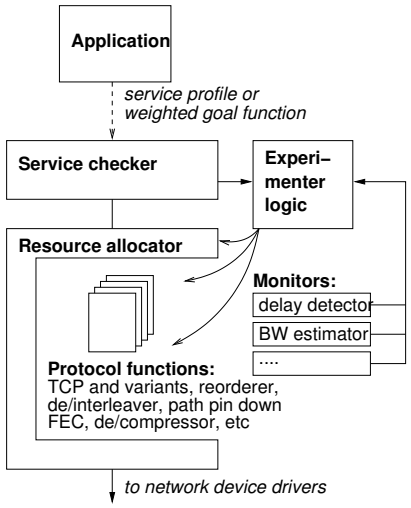
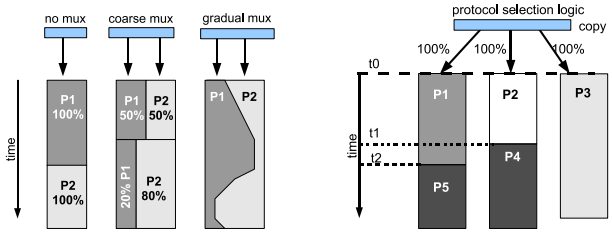



**Figure 1: Architectural framework for online protocol evaluation and switching.**




(a) parallel experiments (b) serial experiments

**Figure 2: Several cases of parallel (a) and serial (b) experiments**

calculations, as well as utility functions which estimate the quality of experience that the user would perceive (e.g. the E-model for VoIP services [10]). The checker can be either passive (just assessing by looking at the flow of data) or active by collaborating with external entities and/or adding its own protocol fields to the application data.

- *Experimenter logic:* Evaluates the running protocols according to the feedback provided by the service checker and monitors, and makes decisions on how to reconfigure the running experiments properly. Possible actions include launch, stop, restart, or reconfigure protocols with different parameters. The application may define the percentage time allowed to perform the experiments. Consequently, the experimenter logic can stop a test which breaks the imposed constraints.

- *Resource allocator:* Executes resource policies such as splitting downstack data streams to two transport paths or imposing rate limitations. To carry out reservations when needed, each protocol function may state its resource requirements.

## 2.2 Runtime inspection and decision

At the core of the framework lies the experimenter decision logic which modifies the configuration of the experiments based on the feedback provided by the monitors and checkers. Applications request services either by choosing the appropriate service checker among predefined categories, or by specifying a target service profile (delay, loss, downtime, bandwidth, each parameter potentially associated with a weight). This reminds us of existing QoS approaches. An important difference is that here the experimenter makes its best to achieve the desired goals by probing multiple alternatives, without an admission control system or similar.

The experimenter starts the service with a first rough choice of at least one production protocol, and optionally one or more experimental ones. The production protocol is typically the most mature among the existing options, or the one judged most suitable to the application, by static knowledge. The experimenter then monitors the "network weather" [24] and can proactively switch protocols, otherwise it periodically evaluates the competition performance and potentially switches to the competitor. The change of a performance variable may be also the trigger of a switch. It is also possible to use two or more variants in parallel and shift traffic ratios smoothly. Fig. 2 shows how parallel and serial experiments can be performed. Fig. 2(a) shows a typical parallel experiment. At time $t_0$ the experiment starts with three protocols in parallel: $P_1$, $P_2$, and $P_3$. Each protocol receives a full copy of the traffic, leading to redundancy which is used for fault tolerance: if one of the protocols misbehaves, the service is still delivered by one of the others. At $t_1$ the experimenter decides that it is better to replace $P_2$ by $P_4$, and at $t_2$ $P_1$ is replaced by $P_5$. Fig. 2(b) shows three possibilities for serial experiments: the first one (left side) is the simplest case, where only one protocol is used at a given time, and the experimenter switches protocols in response to observations from the monitors and the checker. In the second case (middle) a coarse multiplexing of protocols is performed, with part of the traffic sent to one protocol and part to the other. The shares of the different protocols vary in coarse chunks. The third case (right side) shows a fine-grain, gradual multiplexing scheme in which the percentage of traffic that goes to each protocol is adjusted dynamically.

The choice of the observation and switching period is crucial: too short observations do not provide sufficient statistical information, and too long observations provide slow feedback to switching decisions. Furthermore, a too short switching period does not leave sufficient time for new protocols to initialize and adapt to the new environment, does not capitalize on the transition time spent, and could lead to undesirable performance oscillations. In contrast, a too long switching period would result in slow adaptation.

The duration of an individual test can be bound from the application requirements profile, in which the user can specify a maximum threshold of the service degradation during the test stage. This quota can vary depending on the importance and priority of the requested service. The minimum duration of the test must be greater than the protocol initialization delay. The maximum duration will be bound by the total grant for the test stage. To determine the optimal duration for an individual protocol test, it is necessary to measure the time that a protocol needs to stabilize its performance, so that it can be characterized. The stabilization of the protocol operation could be detected by statistical analysis. For instance, the Kolmogorov-Smirnov test could be used to verify whether current and past performance samples belong to the same statistical distribution, assessing that the protocol operation impact remains stable. Similarly, this kind of statistic tools could be employed to determine whether the network conditions change.

# 3. CASE STUDY I: TRANSPORT PROTOCOL SWITCHING

In this section, we describe a series of simulation experiments that demonstrate the usefulness of dynamic protocol switching. We have focused on the transport layer, in the context of a file transfer application, to select the best protocol which best fits the network conditions, as it is described in section 3.1. These experiments will start showing a simple case where the switching between two transport protocols obtain an overall enhanced performance. To this end, we will show two simple reference scenarios in Sect. 3.2. For each scenario, one of the two protocols behaves its best, and the other presents a poor performance. In subsection 3.3 we will present a mixed scenario in which a complete knowledge of the network weather is supposed and where the switching approach is deployed, obtaining an improved performance.

Finally, an experiment to test the feasibility of an auto-

nomic switcher is shown in Sect. 3.4, where a simple automatic procedure triggers online protocol tournaments.

## 3.1 Experimental setup

The file transfer experiment is implemented in the *ns2* simulator [16]. A number of simulations are conducted for several scenarios. The file transfer application can use both TCP [22] and an UDP [23] based transport protocol that we will call *Bulk UDP* (*BUDP*). Both protocols offer the same service: a reliable delivery of data.

The switching approach performed for these experiments is the simple serial switching (refer to Fig. 2). Note that each time that a switch is performed, a *protocol reboot* is needed. This means that, when using TCP, the connection is shut down, thus a reconnection is needed after the switch. This procedure cleans the TCP sending buffer and frees the required resources. Therefore, each switching entails resources allocation, parameter setup and freeing.

Observe that we are working with existing protocols as TCP; we do not intend to reimplement them or change their behavior, achieving thus a versatile framework applicable without modifying existing (and future) protocols. The switching procedure may not know the protocols' operation, treating them as black boxes, using the client feedback of the service and the network impact as indicators.

The BUDP protocol relies on UDP to deliver the data. It has no congestion control scheme, sending the data at a fixed transmission rate. The protocol and the server/client interaction can be described as follows:

1. The client requests the range(s) of bytes that needs to complete the file ($nb_i$ bytes for the cycle number $i$).
2. The client estimates the time ($t$) to get all the packets, which is based on end to end delay ($t_p$), sender rate ($r$), and the number of packets requested ($n$).
3. The server sends the requested bytes in bulk, at a rate $r$, encapsulated in $m$ bytes packets long.
4. If the file has been completely received, (i.e., list of pending bytes is empty) the protocol finishes.
5. When the estimated transmission time $t$ expires, the client *reboots* and goes back to the first step.

It is important to emphasize that we take TCP and BUDP as simple examples to illustrate the autonomic protocol switching concept. Since these two protocols have radically different behaviors, the effects of switching between both can be clearly observed. Our purpose is neither to analyse the performance of TCP, nor to propose the wide use of a naive protocol such as BUDP as an alternative to TCP in specific scenarios. We aim at showing that a dynamic protocol selection framework can provide advantages with respect to static protocol stacks.

## 3.2 Individual protocol performance

In this section, we show the performance of TCP and BUDP individually, in optimum and worst case scenarios for each protocol. While TCP is suitable for the most commonly encountered situations on the Internet, it is not designed to adapt to all cases. For instance, it is well known that TCP has a poor performance under some circumstances such as high packet loss rates and large end to end delays.

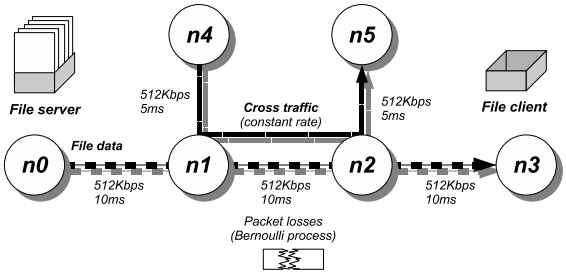



**Figure 3: Simulated scenario.**

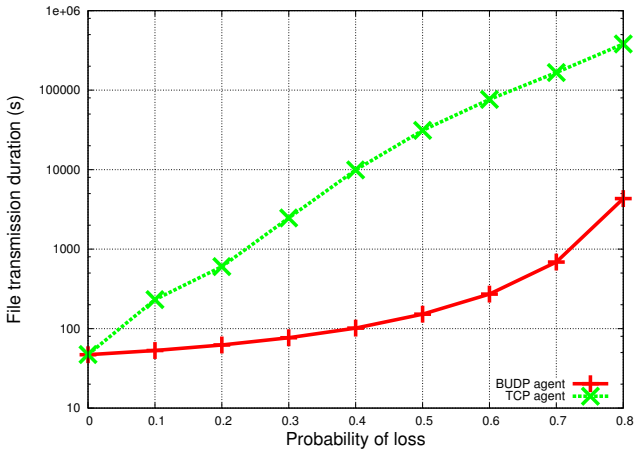



**Figure 4: Transmission duration vs. probability of loss.**

When there is no congestion and a constant amount of bandwidth is available, the use of a protocol with no congestion control and without per packet retransmission can lead to better performance in terms of packet delays. This case can be illustrated with a scenario where there is no congestion, but packet losses occur as shown in Sect. 3.2.1. On the other hand, a congested network is not the best context to deploy a protocol which does not adapt to the available bandwidth, as it is the case of BUDP shown in 3.2.2.

### *3.2.1 Scenario with periodic packet losses*

The scenario shown in Fig. 3 (but without nodes $n4$ and $n5$) is used to describe this case. At node $n0$, a file server will send to the file client (at node $n3$) the requested file in chunks of 500 bytes of data, at a rate of 512 Kbps. The total file size to be sent is $3 \cdot 10^6$ bytes. Each link in the topology has a bandwidth of 512 kbps. The propagation delay for every link is 10 ms. At the intermediate link $n1 - n2$, packets are corrupted with a constant probability $p$ during each simulation. In this scenario, no additional cross traffic will be sent from node $n4$ to $n5$.

For a start, 50 simulations are conducted with TCP as transport agent. Additional 50 simulations of the file transfer using the BUDP transport protocol are executed. For both the TCP and the BUDP protocols executions, Fig. 4 shows the average time of the simulations to transmit completely the file as a function of the probability of loss.

Fig. 4 shows how the required time to deliver the whole file in the TCP case grows exponentially with the probability of loss. This is mainly due to the congestion detection

scheme of TCP: when a retransmission timer expires, the congestion windows decreases to the minimum segment size (MSS).The increase of the window is additive and, therefore, in case of a high probability of loss, the congestion window has the minimum size during almost the whole simulation. This situation can get even worse if we increase the end-to-end delay in the selected topology.

The time to transfer the file for the BUDP protocol is much lower than for TCP: Even for extreme cases of $p = 0.7$, BUDP delivers the file in less than 690 seconds, while TCP needs more than 240 times this value. This performance of BUDP is mainly due to its operation. The sender does not need to wait for positive acknowledges. If the estimated time to get the file (and to send the request for lost packets) is accurate, the resulting time to transmit the file is near the minimum, as well as the required bandwidth and the number of retransmitted packets. Regarding the number of the sent packets, both TCP and BUDP obtain the same results as a function of $p$.

This scenario, as we have seen, is the most suitable for the BUDP protocol. However, let us study a scenario with no link failures and additional traffic sharing the link.

### 3.2.2 *Scenario with congested periods*

In this scenario, depicted in Fig. 3, the only packet losses will be caused by the congestion of a router due to the background crossing traffic. For this experiment, the probability of loss at the $n1 - n2$ link is zero. A constant bitrate traffic traverses link $n1 - n2$ during the whole simulation, consuming thus a percentage of bandwidth. This may cause that the router $n1$ is not able to cope with all the input traffic (background and file data packets), discarding some of them. The size of the cross traffic packets is 125 bytes. This causes the available link bandwidth to decrease, although the file server sending rate is fixed at 512kbps.

For different bandwidth occupation of the cross traffic in the shared link, 50 simulations are conducted for each of the transport protocols. The resulting values for the duration of the transmission vs. the available channel bandwidth percentage can be seen in Fig. 5. Note that the estimated available bandwidth, noted as $B_a$ hereafter, is calculated as the difference of the link capacity and the bitrate of the cross traffic, and, although the calculated available bandwidth is zero, some file packets can be successfully delivered, leading to other traffic packet drops. Comparing the results, no significant differences are found in terms of the average time to transfer the file. However, as it can be seen in Fig. 5, the performance of the BUDP protocol drops when we examine the number of packets sent to complete the file transmission. As we have mentioned before, the lack of an adaptive sending rate strategy or a congestion control procedure makes BUDP unsuitable in case of congestion. In such situation, BUDP will increase the congestion, discarding thus a higher percentage of packets.

On the other hand, TCP behaves fairly in this scenario in terms of bandwidth use. While BUDP sends more than twice the size of the file when additional traffic uses the 80% of the path bandwidth, TCP incurs in a negligible overhead.

## 3.3 Serial protocol Switching performance

The previous section illustrated how each of the studied protocols provides a suitable performance under some given circumstances. In fact, each protocol behaves poorly in the most suitable opponent's scenario. This section shows the maximum improvement that can be achieved with switching under ideal conditions. With this we evaluate the feasibility of switching alone, without taking into account the influence of monitoring, evaluation and switching decision algorithms.

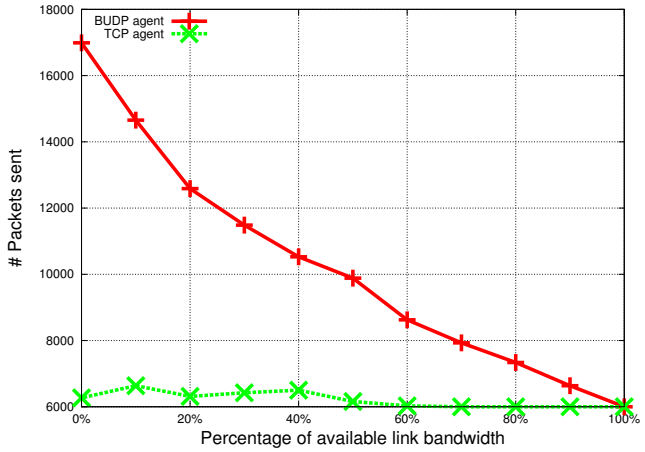



**Figure 5: Number of sent packets vs. percentage of available path bandwidth ($B_a$).**

To illustrate this, the switch will be made following a full knowledge approach, that is to say, the system has a global knowledge of the network conditions and instantly switches to the most adequate protocol when these conditions change. In this case, the file size will be $15 \cdot 10^6$ bytes (30000 packets of 500 bytes each), and the sending rate will be set at 512kbps.

In this scenario, detailed in Fig. 3, both background traffic and losses at link $n1 - n2$ are simulated. During periods 100 to 200 and 300 to 400, cross traffic is injected at a rate of $(1 - B_a) \cdot 512 = 460.8$kbps, so only the 51.2kbps are available during that time ($B_a = 0.1$). During the rest of the simulation, no bandwidth is consumed by other traffic than the generated by the file transfer, but packets are lost at link $n1 - n2$ with a probability $p = 0.3$ instead, as it is labeled in Figs. 6(a,b). The results in Fig. 6(a) and 6(b) show how this switching leads to a solution with the best of all the considered protocols.

During the congested periods (from seconds 100 to 200 and from 300 to 400) the TCP protocol is selected. During the packet loss periods, the BUDP approach is activated instead. As it can be seen, while the pure TCP lasts 11336.27 seconds to deliver the whole file, the switching scheme finishes in 497.6 seconds. In addition, while the pure BUDP protocol sends 56681 packets to get the whole file transferred, resulting thus in an overhead of 88.9%, the switching approach sends 42050 packets (14631 packets less), obtaining an overhead of 40.2% additional packets. In this case, TCP needed to send a 42.9% of additional packets, mainly due to the retransmissions needed during the periods of packet losses with $p = 0.3$. As a result, the switching procedure obtained an overall performance which beats the ones of the TCP and BUDP pure protocols, reducing both the overhead of sent packets and achieving a time performance close to the BUDP approach.

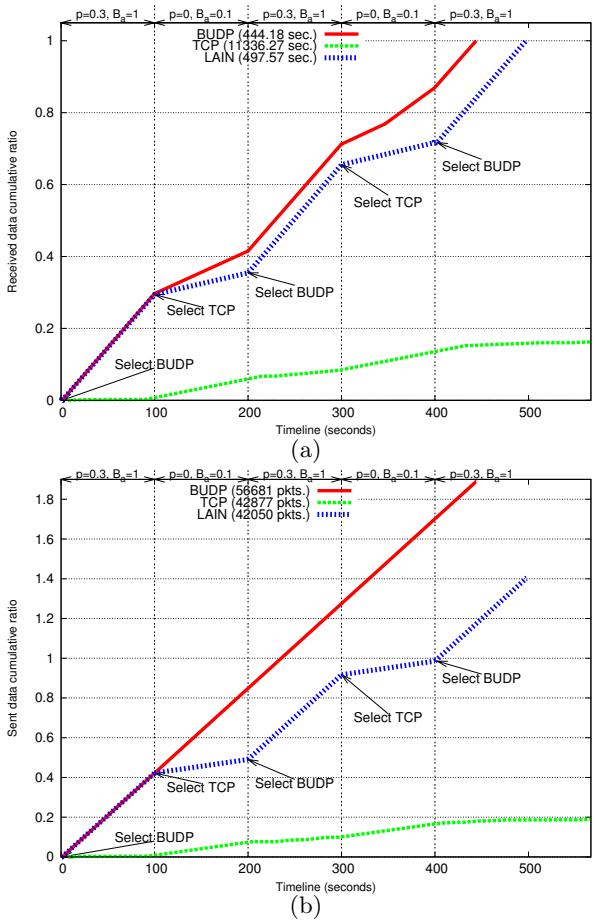



**Figure 6: Fraction of the file received (a) and sent (b) vs. timeline**

## 3.4 Serial online testing performance

In an ideal environment, the system can know which is the most suitable protocol under some given network state, as in the examples of the previous section.

However, it is not possible to have full knowledge about the network weather and the suitability of every new protocol: comprehensive comparisons between other existing protocols should be made, and network parameters have to be precisely identified. Several issues arise in this scheme: how many candidate protocols should compose the tests? In which order? How long should the test last?

The following experiments will show the benefits of an automatic mechanism for triggering the testing stage and switching protocols when needed. For this purpose, we instantiate the elements of the LAIN framework outlined in Sect. 2, to the specific scope of a file transfer service:

**Service checker:** Note that the service implemented in current experiments aims to deliver the entire file as soon as possible, with the minimum bandwidth use. Therefore, the service checker should take into account the duration and the overhead to assess the goodness of the protocol performance. The simple fitness function $\Phi_{ft}$ for the file transfer service used in the experiment is expressed as:

$$\Phi_{ft} = n_r - n_l \quad (1)$$

where $n_r$ is the number of successfully received packets (replicated packets are not considered), and $n_l$ is the number of detected packet losses, including the retransmitted lost packets. Note that $\Phi_{ft}$ could become negative in cases in which a selection incurs in a high percentage of loss, such as using BUDP in a heavy congested scenario, which are counterproductive scenarios that must be penalized.

**Monitor helper:** In order to check whether the network conditions change, periodic probing packets, which generate a negligible bandwidth use (10 probing packets per second, consuming only 0.4 kbps), are sent by the server side of the monitor. When the client detects a change, it notifies the server of the new network state.

The test procedure will be triggered when the network monitor notices a change. To this end, the current and the previous stable states are characterized by the Jacobson expression proposed for the estimation of the round trip time ($rtt$) in TCP [12]. The network monitor assesses that the network weather (the packets delay in this case) is stable when the condition 2 is verified:

$$\mu_{t_{i-1}} - 2 \cdot \sigma_{t_{i-1}} < \mu_{t_i} < \mu_{t_{i-1}} + 2 \cdot \sigma_{t_{i-1}} \quad (2)$$

where $\mu_{t_{i-1}}$ and $\sigma_{t_{i-1}}$ represent the smoothened average and the variance of the previous stable state of the network respectively, and $\mu_{t_i}$ characterized the current average delay. These values are updated with every probe packet as shown in equations 3 and 4:

$$\sigma_{t_i} = (1-\beta) \cdot \sigma_{t_{i-1}} + \beta \cdot |\mu_{t_{i-1}} - (t_{now} - t_{sent})| \quad (3)$$

$$\mu_{t_i} = (1-\alpha) \cdot \mu_{t_{i-1}} + \alpha \cdot (t_{now} - t_{sent})) \quad (4)$$

Where $t_{now}$ is the packet arrival time, and $t_{sent}$ is the sending timestamp of the packet. $\alpha = \frac{1}{4}$ and $\beta = \frac{1}{8}$ as recommended in [12].

**Experimenter:** In case a change is detected (condition 2 is not verified), the test stage is triggered.

As a result, shown in Fig. 7(a) and Fig. 7(b), we obtain a plot which outperforms both BUDP and TCP. As in previous scenarios, the time to transmit for LAIN is around 23.66 times lesser than TCP's, and the number of sent packets differs in 7832 packets to the BUDP protocol.

However, the use of the network monitor in order to trigger the tests leads to tests rounds which are not necessary, as it can be seen in Fig. 7 at second 141.1, where a test round is activated (the unnecessary tests and switch are marked with an asterisk). This shows that the network monitor and the experimenter decision maker have to be carefully designed in order to avoid unnecessary tests.

# 4. CASE STUDY II: SWITCHING INTERLEAVERS IN AUDIO STREAMS

Although the proposed framework is mainly targeted to autonomous systems in which protocol switching is a long term target, in this section we present a scenario which shows how the switching procedure can be applied to applications which need to adapt in a short period of time.

Voice over IP (VoIP) is an example of this kind of applications. VoIP traffic is characterized mainly by its tight time restrictions (packets' end-to-end delay must be lower than

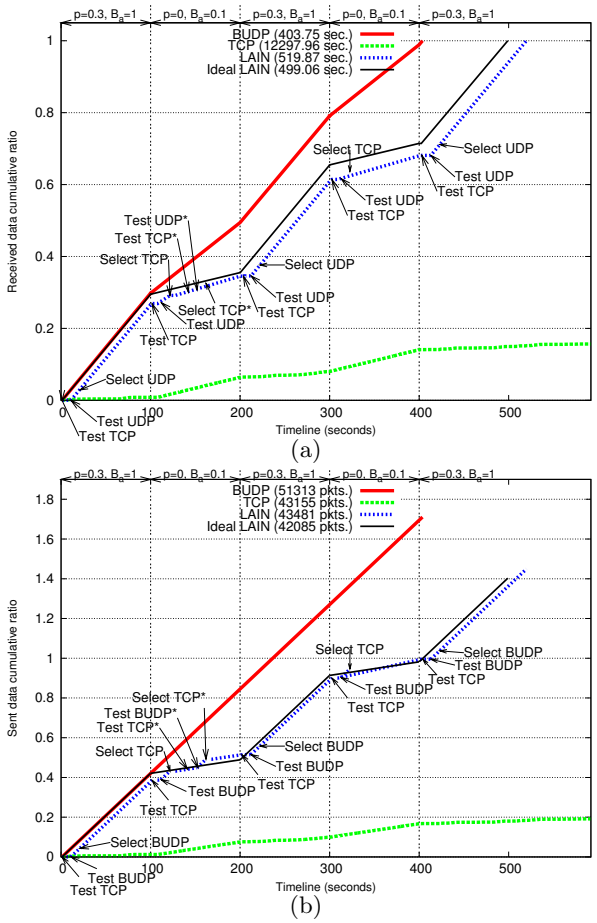



**Figure 7: Fraction of the file received (a) and sent (b) vs. timeline**

a threshold $D_{max} = 300ms$) and its tolerance to low packet loss rates [20]. On the contrary, consecutive packet losses lead to a rapid degradation of the quality of the perceived voice stream [13].

Several approaches have been proposed to cope with this problem, providing techniques such as the so called *packet interleaving* [20]. Basically, the interleaver scrambles the packets obtaining thus a stream in which bursts of errors become individual losses when the stream is ordered at the receiver. Its main drawback is the additional incurred delay.

The so called *multiflow block interleaver* type $II(n_f, s)$ [18] is a special kind of block interleaver which takes packets from several VoIP sessions and outputs an aggregate stream with the desired isolation capabilities, introducing the minimum theorical delay.

In addition to the number of available flows ($n_f$) to perform the interleaving, the device needs as input the target burst length ($s$), that is to say, the maximum burst length that it will isolate. For a given pair of $n_f$ and $s$, the resulting device provides the minimum delay interleaving with a maximum of $d_{n_f,s}$ seconds of end-to-end delay. Such delay is not monotonically decreasing with the target burst length.

Instead, the $d_{n_f,s}$ delay introduced by the interleaving follows a non monotonic trend. The calculation of that delay is shown in detail in [18]. In our experiments we will activate an interleaver that has to switch the $s$ target parameter to the actual burst lengths in order to optimize the resulting packet loss distributions and the incurred delay.

Such selection has to be made regarding that the maximum delay must be lower that $D_{max}$.

## 4.1 Framework elements

Several LAIN framework elements had to be instantiated for our simulations.

### 4.1.1 *Monitor helpers*

The implemented monitor checks whether the packet loss pattern changes or keeps stable. To this end, the monitor at the receiver side logs the burst distribution of the transmission path by means of maintaining a histogram of the experienced packet losses.

Periodically, the loss monitor sends the updated statistics of the link loss pattern and the average packet delay ($d$) to the experimenter. More precisely, the receiver side of the monitor sends back to the source the $80^{th}$ percentile of the receiver cumulative distribution function (CDF) of the burst length. The meaning of this value is that the 80% of the experimented bursts of losses are equal or lower than this score, noted as $b_{80^{th}}$ hereafter.

Finally, the histogram is reset each time that a switch is performed or a test round is triggered.

### 4.1.2 *Service Checker*

The service checker for this experiment uses a fitness function $\Phi_{d,b_{80^{th}}}$ which considers the average burst length and the experimented end-to-end delay, (eq. 5):

$$\Phi_{d,b_{80^{th}}} = \frac{1}{b_{80^{th}}} \cdot \frac{D_{max} - d}{D_{max}} \qquad (5)$$

Where $d$ is the average packet delay, $D_{max}$ is the maximum tolerated end-to-end delay (300 ms in our case), and $b_{80^{th}}$ is the $80^{th}$ percentile of the experimented burst length CDF.

A candidate performs better than another if its fitness score is greater than the other candidate. The fitness score is thus designed to highlight those results with a final short average burst length and shorter delays.

### 4.1.3 *Experimenter logic*

Each time the network loss distribution changes, the test tournament is triggered. Also, periodic tests tournaments are held each 75 seconds. At the beginning of the test, a number of interleaver candidates are generated. Such candidates must verify that their maximum delay is lower than the maximum threshold $D_{max}$. Their input parameters are $n_f = 4$ and $s$ (which is chosen to be close to the $b_{80^{th}}$ value reported by the monitor helper).

In our simulations, only the 3 candidates (out of 21) with the shortest distance to the $b_{80^{th}}$ value are tested. They are activated for a period of 5 seconds, registering their fitness scores. After the test round, the one with the best fitness value is selected, and the production interleaver is substituted by switching.

A test round is also triggered when a pattern change is detected. A change of network's conditions stability is detected when one of the conditions expressed in Eq. 6 below is verified:

$$\lceil b_{80^{th},i-1} + 1.5 \rceil < b_{80^{th},i} < \lfloor b_{80^{th},i-1} - 1.5 \rfloor \qquad (6)$$

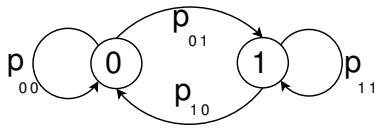



**Figure 8: Gilbert model**

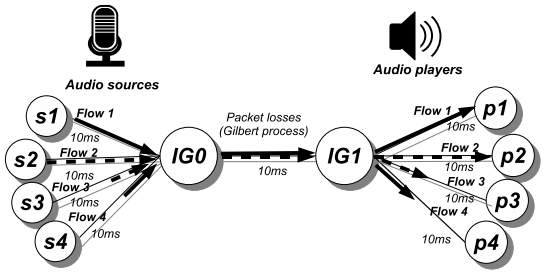



**Figure 9: Interleaving scenario topology**

where $b_{80^{th},i}$ is the most recently reported value, and $b_{80^{th},i-1}$ is the last stable period value. The current $b_{80^{th},i}$ is recalculated after every update report by the smoothing average detailed in the equation 7:

$$b_{80^{th},i-1} = 0.9 \cdot b_{80^{th},i-1} + 0.1 \cdot b_{80^{th},i} \quad (7)$$

It is worth noting that the number of candidates, their selection order, the period of the per protocol tests and the frequency of the tournament test are key parameters which have to be carefully selected. Due to the exploratory nature of our experiments, those parameters have been selected to be simple, and therefore not necessarily optimized.

## 4.2 Experimental setup

This scenario consists of 4 audio sources which send packets through a common path. The path is subject to a changing loss pattern generated by a *Gilbert model* [9]. The Gilbert model consists of a 2 state Markov chain. One of the states represents the error-free period (labeled with "0"), and the other represents the burst of losses (label "1"). Fig. 8 depicts the two-state Gilbert model.

The probabilities of transition from the error-free state to the error state ($p_{01}$) and from the error state to the error-free node ($p_{10}$) characterize the model.

Fig. 9 shows the simulated topology. It consists of 4 source nodes which originate the audio streams, 4 sink nodes which represent the listeners, and 2 common nodes, labeled $IG0$ and $IG1$, which are connected to all the streams via the common link where errors occur. The source nodes produce audio packets with a sending rate of 50 packets per second. The so called *Type* $II(4, s)$ *Multiflow block Interleaver* is located at node $IG0$.

In order to assess the perceived quality of voice transmissions, the *E-model* has been proposed by the ITU-T [10]. The E-model provides an estimation of the Mean Opinion Score (MOS) of the voice quality that real users would assess in a MOS test. The MOS scale ranges from 1 ("bad quality") to 5 ("excellent quality") [11]. A drawback of the E-model is that, although it estimates the voice quality, it does not completely model all the involved perceptual factors. In fact, the E-model does not capture the bursty behavior of losses (it only uses the overall probability of loss), and therefore it is not sufficient to assess the performance of the interleaving scheme. To provide a complete quality score which takes into account all the impairments which affect the perceived quality, the average resulting burst length $b_L$ and the average packet delay $\mu_d$, key parameters to assess a VoIP stream, will also be provided in our evaluations.

| Scheme | s | MOS | $\mathbf{b_L}$ | $\mu_d$ (sec.) |
|---|---|---|---|---|
| LAIN | $[4-13]$ | **2.33** | **2.87** | **0.099** |
| Fixed | 5 | 2.35 | 3.31 | 0.080 |
| | **6** | **2.31** | **2.85** | **0.120** |
| | 7 | 2.29 | 2.58 | 0.135 |
| | 8 | 2.32 | 2.38 | 0.110 |
| | 9 | 2.01 | 2.24 | 0.211 |
| | 10 | 1.59 | 2.11 | 0.280 |
| | 12 | 1.65 | 1.94 | 0.270 |

**Table 1: Results for changing network conditions**

## 4.3 Serial on-line testing performance

The evaluation is held in a scenario where the error patterns change in time. The switching procedure has to result in the adaptation of the interleaver to the given conditions, in order to obtain better performance (and perceived quality) than in a classical approach.

The evaluation will be provided with the MOS score obtained by the application of the E-model and the average per packet delays and resulting burst lengths.

In the experiment with the fixed $TypeII(4, s)$ interleaver, the target burst length ($s$) is set at the beginning of the simulations. In order to compare the static strategy with the adaptive approach, several simulations with different values for $s$ from 5 through 13, which are the possible values within the time restrictions, are conducted to find the best result with a static strategy.

The network loss properties change during the voice over IP session. More precisely, from seconds 0.001 to 500 the Gilbert model is characterized by $p_{01} = 0.03571$ and $p_{10} = 0.08334$, resulting in an overall probability of loss $p = 0.3$ and an average burst length $b_L = 12$. From seconds 500 to 1000, $p_{01} = 0.0125$, and $p_{10} = 0.05$, resulting in $p = 0.2$ and $b_L = 20$.

Table 1 presents the results for these simulations. The MOS score obtained by the E-model, the average burst length $b_L$ and the average per packet delay $\mu_d$ are shown for the different possible $s$ values verifying the $D_{max}$ deadline constraint. To obtain the same MOS score and $b_L$ value as the LAIN approach with the fixed scheme, at least $s = 6$ must be chosen. However, the average introduced delay $\mu_d$ is 0.019 seconds greater. To obtain a lower delay, $s = 5$ could be chosen for the fixed approach, but the average burst length $b_L = 3.31$ exceeds in 0.44 the value obtained by our LAIN approach. Note that for a high probability of loss (as in this case, $p = 0.3$), even small differences in the average burst length have a great impact on the perceived quality.

# 5. DISCUSSION AND RELATED WORK

In the previous sections we have demonstrated dynamic protocol switching at the transport level for an elastic as well as an inelastic application. In both cases, we see that observing and switching protocols dynamically can be beneficial

in many situations, even with the very simple monitoring and selection strategies implemented. Monitoring capability and appropriate switching logic are important issues which have to be further investigated. In this section, we discuss more research questions and related work, and present an outlook to the future with automatic protocol evolution.

## 5.1 Protocol Variety

Our framework assumes that plentiful protocol choice is available. Several approaches have provided solution to this question, for example x-kernel, ComScript, active networking, protocol toolkits like Click, and (active) middleware. As a recent example, a dataflow system is presented in [4], in which protocols can be dynamically assembled out of reusable building blocks for peer-to-peer overlay networks. More related to TCP we point out the work of Patel et al. who propose an extension XTP and later a protocol upgrade system STP system [19] (Self-Spreading Transport Protocols) where many protocols become potential candidates to be run, hence selection among them is necessary. STP includes a policy manager that rejects or accepts protocols based on their origin (e.g. trusted software vendor) or the history of their behavior. However, no scheme is presented on how to obtain such a history. The design of policies is recognized as important but left for future work.

We assert that there is no shortage in protocol composition and deployment mechanisms but that there is a lack in policies and algorithms to automate the choice of elements to compose and deploy, and to bring this choice into productive use.

An algorithm to perform such choice is proposed in the context of autonomic computing [26], to select automatically among competing components based on their workload characteristics. In [6], the authors propose an algorithm to select the most appropriate web service from a set of candidates. In both cases, accurate performance profiles are key parameters in the switching decision.

The LAIN framework differs from related work mainly in that it does not assume full knowledge about protocol performance or behavior, nor a full network characterization. Therefore, it is inherently more robust to protocol misbehavior or inadequacy. This is the reason why a previous protocol profiling in every possible scenario can not be used. On the other hand, our framework requires timely protocol switching under reliable monitoring information. An algorithm to perform the optimum switch is one of our key future work objectives.

## 5.2 Automatic Protocol Evolution

The framework proposed is part of a wider research agenda on how to automate the evolution of the network and its protocols. The fundamental question is how to create new protocols automatically based on existing, working ones. An equally important question, and focus of this paper, is how to evaluate the suitability of the newly produced protocols for the intended task.

Synthesizing new code can be approached via deterministic, formal methods or via non-deterministic trial-and-error methods such as genetic programming. The formal method approach, albeit promising, is difficult to transpose to an online setting and to generalize to large and complex distributed systems. The non-deterministic path seems easier, however, the candidate solutions produced can not be guaranteed to provide the intended service before being extensively tested for fitness. Therefore, a resilient run-time evaluation method for these cases must be found.

An adaptation scheme based on genetic algorithms is proposed in [17]. Candidate solutions are evaluated dynamically in the real world. The problem of online evaluation under environmental changes in [17] is similar to ours. However, we focus on the evaluation and selection of existing solutions (protocols), while [17] focuses on generating combined solutions of online and offline genetic algorithms.

In a previous work [25], we used genetic programming to recombine and mutate existing protocols in order to evolve new solutions in a simulated online setting. The idea of evaluating and selecting protocols online is presented there, and further developed in the present paper to cover concrete, real-world protocols.

We believe that this field of research could give rise to a new era of self-sustaining, almost organic systems whose constituent parts could suffer any sort of damage or attack, and the system would be able to detect and restore the affected parts in a self-healing manner.

## 5.3 Interface Issues

Since the LAIN framework envisages completely new protocols being added dynamically to the system, the definition of a flexible and extensible Application Programming Interface (API) is a must to let new and existing protocols interoperate, and to allow the applications to specify their needs and preferences in a flexible and protocol-agnostic way.

There is a growing tendency nowadays towards ontology-oriented interfaces, in which components' interfaces are described through standardized languages such as IDL [15], WSDL [3] or OWL [5]. In this context, reasoning tools are proposed to find a best match between application requirements and the potential services satisfying them.

As a first step, service description interfaces could be used in LAIN to help screening protocol candidates before testing them. Only those descriptions satisfying the application specifications would be retained. However, descriptions do not exempt the framework from performing regular intensive tests to evaluate the actual behavior of a protocol, which in many situations might not comply to the description.

In the long term, however, we believe that such predefined descriptions do not offer an ideal solution. In Genetic Programming, researchers are studying how modules and their interfaces could emerge via evolution [27]. Stretching this idea far into the future of distributed software systems, one could imagine the notion of well-defined, independent protocol entities yielding to smaller, atomic interacting units that may form short-lived clusters to solve a given problem or to provide a given service, then dissolve when no longer needed. A new kind of emergent interfaces would be needed for such systems, with all their implications in terms of how to control the system and specify a desired behavior without preprogramming it.

# 6. CONCLUSIONS

We have presented a framework for dynamic protocol evaluation and selection, with initial feasibility experiments on switching protocols online. To this end, case studies covering two categories of applications with different characteristics have been shown: an elastic file transfer application and an inelastic real-time audio streaming application. The bene-

fits and trade-offs involved in protocol switching have been exposed for each case.

As in any emerging field, many issues remain to be solved. The design of a service checker for protocols may be quite hard. Although schemes to differentiate normal from abnormal protocol behavior have been proposed [8], the functionality check of a service from a high level description is still mostly unsolved.

Fitness evaluation requires continuous tests: their duration and resources consumed should be carefully assessed. Moreover, tests may affect network behavior, making the comparison between protocols difficult.

The appropriate monitoring and switching logic needs to be carefully studied in order to detect the correlation between actions and events, which is a classical problem in feedback systems needed for continuous evaluation.

The coordination (signaling) needed is also a matter of concern: in a distributed system, fitness should be reported by the remote side in an accurate and trustworthy manner; the process of switching protocols must happen in a coordinated way such that all end points use compatible protocols.

Another issue of concern is how to ensure fairness among competing sessions when diverse protocols may be selected at any time. The classical solution is to impose a good behavior, such as TCP-friendly, or a fair resource allocation scheme. A different solution could be to allow the network to do also experiments against the users for ensuring convergence to a fair allocation.

Although challenging, this seems a promising research direction, ultimately leading to future networks in which protocols can be synthesized automatically, which evolve to match new needs, and heal themselves against disruptions.

## 7. ACKNOWLEDGEMENTS

This work has been partially supported by the Spanish Government through MEC (Project TSI2005-08145-C02-02, FEDER funds 70%) and by the European Union (IST FET Project BIONETS FP6-027748). Special thanks to Christophe Jelger for the fruitful discussions, to María Elena Rodríguez for the final revision, and to the anonymous reviewers for their helpful comments.

## 8. REFERENCES

[1] S. A. Baset and H. Schulzrinne. An analysis of the skype peer-to-peer internel telephony protocol. Technical Report CUCS-039-04, Dep. of C.S., Columbia University, New York, Sep 2004.

[2] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox. Microreboot - A Technique for Cheap Recovery. In *Proc. 6th Symposium on Operating Systems Design and Implementation (OSDI)*, San Francisco, CA, USA, Dec. 2004.

[3] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL) 1.1. ., 2001.

[4] T. Condie, J. M. Hellerstein, P. Maniatis, S. Rhea, and T. Roscoe. Finally, a Use for Componentized Transport Protocols. In *Fourth Workshop on Hot Topics in Networks (HotNets-IV)*, College Park, Maryland, USA, Nov. 2005.

[5] M. Dean and G. Schreiber. OWL web ontology language reference. W3C recommendation, W3C, Feb. 2004.

[6] A. Diaconescu and J. Murphy. Automating the performance management of component-based enterprise systems through the use of redundancy. In *ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 44–53, New York, NY, USA, 2005. ACM.

[7] N. Dukkipati, Y. Ganjali, and R. Zhang-Shen. Typical versus Worst Case Design in Networking. In *Fourth Workshop on Hot Topics in Networks (HotNets-IV)*, College Park, Maryland, USA, Nov. 2005.

[8] J. M. Estévez-Tapiador, P. García-Teodoro, and J. E. Díaz-Verdejo. Measuring normality in http traffic for anomaly-based intrusion detection. *Computer Networks*, 45(2):175–193, 2004.

[9] E. Gilbert. Capacity of a burst-noise channel. Technical Report 39, Bell Syst., 1960.

[10] ITU-T Recommendation G.107. The Emodel, a computational model for use in transmission planning, July 2002.

[11] ITU-T Recommendation P.800, Methods for Subjective Determination of Transmission Quality, Aug. 1996.

[12] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM '88*, pages 314–329, Stanford, CA, Aug. 1988.

[13] W. Jiang and H. Schulzrinne. Comparison and optimization of packet loss repair methods on voip perceived quality under bursty loss. In *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 73–81, New York, NY, USA, 2002. ACM Press.

[14] C. Knutson, H. Duffin, J. Brown, S. Barnes, and R. Woodings. Dynamic autonomous transport selection in heterogeneous wireless environments. *Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE*, 2:689–694 Vol.2, 21-25 March 2004.

[15] D. A. Lamb. Idl: sharing intermediate representations. *ACM Trans. Program. Lang. Syst.*, 9(3):297–318, 1987.

[16] S. McCanne and S. Floyd. ns2 network simulator.

[17] N. Mori and K. Matsumoto. Adaptation to a Dynamical Environment by Means of the Environment Identifying Genetic Algorithm. *Congress on Evolutionary Computation (CEC '03)*, 3:1626–1631, Dec. 2003.

[18] J. R. Munoz and J. L. Soler. Low delay multiflow block interleavers for real-time audio. *Lecture Notes in Computer Science*, 3420/2005, 2005.

[19] P. Patel, A. Whitaker, D. Wetherall, J. Lepreau, and T. Stack. Upgrading Transport Protocols using Untrusted Mobile Code. In *Proc. 19th ACM Symposium on Operating System Principles*, Oct. 2003.

[20] C. Perkins, O. Hodson, and V. Hardman. A survey of packet loss recovery techniques for streaming audio. *IEEE Network*, 12:40–48, Sep/Oct 1998.

[21] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. RFC 3489 - STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs), Mar. 2003.

[22] Transmission Control Protocol (RFC 793), Sept. 1981.

[23] User Datagram Protocol (RFC 768), Aug. 1980.

[24] R. Wolski. Dynamically Forecasting Network Performance Using the Network Weather Service. *Journal of Cluster Computing*, 1:119–132, Jan. 1998.

[25] L. Yamamoto and C. Tschudin. Experiments on the Automatic Evolution of Protocols using Genetic Programming. In *Proc. 2nd Workshop on Autonomic Communication (WAC)*, Athens, Greece, Oct. 2005.

[26] D. M. Yellin. Competitive algorithms for the dynamic selection of component implementations. *IBM Systems Journal*, 42(1):85 – 97, Jan. 2003.

[27] T. Yu. Hierarchical processing for evolving recursive and modular programs using higher-order functions and lambda abstraction. *Genetic Programming and Evolvable Machines*, 2(4):345–380, 2001.