Examination of Free Capacity Based Load Sharing

Attila Takács, András Császár, Róbert Szabó, Tamás Henk
High Speed Networks Laboratory
Department of Telecommunications and Media Informatics
Budapest University of Technology and Economics
Magyar Tudósok Körútja 2., Budapest, Hungary, H-1117
E-mail:{takacs,andras.csaszar,robert.szabo,henk}@tmit.bme.hu

Abstract

Sophisticated traffic engineering (TE) methods are currently under development. The price for broad TE functionality is high, especially in complexity and scalability. Moreover, there are several specialised networks where simpler TE mechanisms are required because of computational or scalability limitations. Such a simpler TE solution is load sharing. Currently, the performance of load sharing is far less exploited than its real potential. This is due to the fact that thus far no sophisticated load sharing architectures have been defined and that the instability issues of dynamic load sharing have not fully been examined and solved. With our paper we aim to extend the understanding of instability issues of dynamic load sharing solutions.

KEY WORDS
Traffic Engineering, Dynamic Load Sharing, Instability

1 Introduction

The different service requirements posed by network users of the new millennium challenge both network providers and architectures. The mechanisms to support the quality needs of emerging services while utilising resources economically and efficiently are identified as traffic engineering (TE) [1]. Currently, TE research is focused on Multi-Protocol Label Switching (MPLS) [2]. After MPLS appeared in the networking arena, it became shortly the ultimate TE solution. The popularity of MPLS is based on the broad range of TE solutions it is able to provide. Although sophisticated TE mechanisms are under development, currently only a simple TE solution – load sharing – is used.

The main stream of actual load sharing research is focusing on how to derive individual OSPF or IS-IS link costs to realize load sharing with shortest path routing. [3][4] This is practically important, as this kind of load sharing would not require new hardware to be deployed. This research implicitly considers load sharing only as a transient solution as long as MPLS and with it sophisticated TE solutions are deployed. However, there are a number of scenarios where load sharing is the right TE solution because of, e.g., its simplicity. For example, in IP based Radio Access Networks (RANs) where several thousand routers are interconnected with a meshed topology a complex small/middle time scale TE solution is not applicable due the high computational costs. Because of such specialised networks the development of sophisticated load sharing architectures and solutions must be considered. Additionally, as load sharing solutions should be able to quickly react to changes in the network, the understanding of dynamism in load sharing must be improved since the investigation of dynamism came short in recent literature. Questions about dynamism, its problems and some solutions last came up for animated discussion during the investigation of the Optimised Multi-Path (OMP) framework [5]. Unfortunately, OMP has not became widely used as it could solve its instability issues only at the price of slow adaptation.

Two tasks must be addressed to make dynamic load sharing attractive for networks based either on MPLS or solely IP. i) An architectural framework is needed, which builds a simple but solid basis for load sharing supporting the requirements to maintain packet ordering and to reduce the possibility of oscillation. Such a framework, called Core State Limited Load Sharing (CSLLS), has been presented in [6]. ii) As dynamism is mainly a property of the utilised load sharing algorithms, these must be analytically investigated to gain better understanding about instability and oscillation. As a result guidelines must be given to prevent these undesired phenomenon.

The paper is organised as follows. In Section 2 we shortly introduce the CSLLS framework. In Section 3 we investigate the load sharing algorithms to characterise instability. In this section we also present solutions to prevent oscillation. Numerical results are presented in Section 4. A conclusion is drawn in Section 5.

2 The CSLLS framework

The acronym CSLLS stays for Core State Limited Load Sharing. The architecture is designed to overcome the weaknesses of previous attempts and incorporates all the necessary features and flexibility to represent a solid base for a *thrifty* traffic engineering solution.

Generally, the problem of routers with load sharing is the use of multiple paths towards the same destination. The forwarding table must be extended to accommodate the additional information of multiple paths. CSLLS proposes an

extension which not only allows the use of multiple paths with arbitrary load splitting among them, but also supports dynamism. For every destination a vector is given where every element is assigned to an outgoing link (or port). The elements of the vector are threshold values, which are interpreted as follows. The probability of routing a flow over the ith outgoing link is the difference of the vector values of the ith and the $i-1$th element. The probabilities are stored as thresholds to make it easier to select the appropriate outgoing link for an incoming packet, as the packet is carrying a value $v \in [0,1]$ (or this is calculated based on a hashing scheme), which is destined to select the path of the packet. With the use of a vector instead of a single value for the forwarding table, CSLLS solves the basic problem of load sharing. Though, CSLLS goes further and instead a vector it allows to use a matrix consisting of vectors described above. This extension makes it possible to use different load splits for different time-stamps. This helps to preserve the paths of ongoing flows while new flows can be routed differently, to react to changing network conditions. To utilise this possibility each flow (more precisely every packet of the flow) is "tagged" with a time-stamp value which remains constant through the lifetime of the flow. In particular this tagging hashes the flows according to their arrival times at the ingress node. Hence, the path of a flow depends on the arrival time and a random value (or an IP header hash). As the reader already noticed, we are likely to speak about load sharing of flows rather than packets because the requirement of preserving packet ordering makes it more adequate.

Adaptation to load changes is achieved by altering a vector in the forwarding matrix. If we want to guarantee that the path of flows will never change we would need – in worst case, if the lifetime of flows is not limited – an infinitely large matrix, to store all previous split changes. However, we would like to have an architecture which is scalable and needs sparse states in the routers. The need to have as few states as possible and the need to preserve path integrity contradicts each other. A good trade-off must be found so that we can define the number of vectors needed (N) and the time intervals (ΔT_i) these vectors are updated. After we reach the last vector, we will continue updating with the first one in a Round-Robin fashion, so if the intervals are constant then the path integrity of a flow is guaranteed $(N-1)\cdot\Delta T$ long time. The updates need not be periodical they could be triggered by predefined network load changes. In the rest of the paper to simplify the discussion we suppose that the updates are periodical with the intervals of ΔT. The time-stamp tagging of arriving flows must conform the intervals, hence if we updated the next vector, we must tag all the new flows with the identifier of the updated vector. If periodical updates are made, then some kind of synchronisation is required. If updates are triggered, then all nodes must be able to catch the signals to align the tagging procedure. Of course, the transients of vector updates and tagging align must be solved acceptable fast to avoid inconsistent packet forwarding.

3 Load sharing

Now that we have an architecture that supports sophisticated network load distribution, we have to construct an algorithm that will set the load sharing ratios among different paths or next-hops. CSLLS naturally offers the usage of free capacity information to set the load splits. This is due the fact that previous flows tagged with previous time-stamps will not be re-routed, so the network virtually has to distribute the available capacity to the flows arriving during the next ΔT interval. This way, if the load sharing algorithm receives link state information (e.g., link utilisations and link capacities) from the entire network as input, a graph can be constructed in which a proper algorithm can look at the paths between source-destination node pairs and calculate the available bandwidths for each.

With CSLLS, the sharing ratios of the alternative paths are set according to the assigned weight of each path. In the remainder of this section we will consider what problems may occur with weight proportional load sharing and we will investigate whether it is the best if the weight of a path is directly the free bandwidth on that path or we can do better than that.

As examined in [7], different hashing strategies produce differently "good" random numbers that may result a difference between the desired and the realised load sharing ratios. Specifically, if the hash values are not uniformly distributed in $[0;1]$, there could be great differences between the desired and realised load sharing ratios. The second problem load sharing has to face is that let the weights and load sharing ratios be set whatever optimal, only a finite number of new flows arrive. Load sharing realized with a finite number of random values only approximates the preferred load distribution (supposing two alternative paths, the difference can be described mathematically with a binomial distribution). Fig. 1 shows the difference between the weights set by CSLLS and the realized load distribution in our simulations. The smaller ΔT is, the less flows arrive in the intervals, and the higher will be the mismatch. In a highly loaded environment this may even lead to slightly overload some paths while other paths might have some free capacity left. Luckily, the dynamism of CSLLS compensates these aberrations because the free capacities at the end of the ΔT interval will reflect these differences. The third problem is about load sharing decisions of routers affecting each other. This problem requires further research, in this paper we examine load sharing from the point-of-view of a single router that can balance the load between multiple paths.

Now, let our attention turn to the question of assigning weights to the paths. In the following let w_i denote the weight of path i, let f_i denote the free capacity of path i. As of now it seems plausible to set up the weights in direct linear proportion to the free capacity: $w_i = f_i$. This algorithm will be denoted as LIN in the followings.

If we suppose that the newly arriving traffic is less than or equal to the sum of the available capacity on all

paths – meaning that ideally the load can be distributed without blocking –, then this kind of weight setup naturally gives the highest probability of avoiding blocking. At least we cannot say anything better than that.

If we suppose that the newly arriving traffic during a ΔT interval is more than the summarised free capacity on the alternative paths and existing flows do not leave, then the network is obviously overloaded. In this case, LIN blocks some flows but again, any other weight setup would block the same amount or more flows.

In a real environment however, existing flows also leave the network and release bandwidth continuously. If we add this "about-to-be-released" capacity to the current free capacity, we can route more flows without blocking. Let us show you an example. Given two alternative next-hops, through each 1MBps capacity path is leading to the destination. At an update, the free capacities of these paths are 1kBps and 2kBps. Example (a): If during the next ΔT interval, three 1kBps sessions arrive then, assuming a good hash distribution, the $\{\frac{1}{3};\frac{2}{3}\}$ weights mean the one and only perfect configuration. In example (b) on the other hand, if 300 pieces 1kBps flows arrive during a ΔT interval, 100kBps will be directed onto the first path and 200kBps onto the second. Both paths will be overloaded, hence there is no chance to properly route the new flows. However, this is only true if on-going flows do not terminate and do not release resources when leaving the network. But generally, they do. If (example (c)) 150 calls leave each path, the $\{\frac{1}{3};\frac{2}{3}\}$ weighting would drop 50 newly arriving sessions, but at the end of the interval 50kBps would be free on the under-weighted path. Moreover, the ratio of the free capacities at the next update would be high, and this time the previously under-weighted path would receive too much traffic. The result is that the sharing ratios begin to oscillate. If this fluctuation affects many flows like in the above example, the result is unsatisfying network utilisation and higher blocking probability than necessary.

Considering a busy, high capacity network with many flows, one can assume (or cannot assume more than) that both session arrivals and terminations are uniformly distributed during a ΔT period, so released resources will effectively be available for arriving sessions. If we had estimated session leavings for 100-100 flows from each path, then the $\{\frac{101}{203};\frac{102}{203}\}$ weights would yield a good configuration for example (c).

The amount of leaving sessions is a positive, monotony increasing function of time. If information about mean remaining holding times of sessions (denoted by h) are known, one can simply estimate the available capacity during the next interval for next hop i as:

$$\hat{f}_i = f_i + MAX(1; \frac{\Delta T}{h}) \cdot u_i, \tag{1}$$

where u is the link utilization. (We will refer this function as ERR-Mean, where ERR is the shortcut of Expected Resource Release.)

Better estimates are also possible. For example, we simulated call holding times with an exponential distribution with parameter λ. Since the exponential distribution has a property of being perpetual youth, the same exponential distribution can be used for estimating the remaining holding time of sessions. Let $F(x) = 1 - e^{-\lambda \cdot x}$ denote the distribution function which is the probability that the holding time will be less than x. With this the estimated available capacity could be:

$$\hat{f}_i = f_i + F(\Delta T) \cdot u_i. \tag{2}$$

(We will refer this function as ERR-Distribution.)

If we know that resources will be released but we cannot or do not want to say anything more, we can still perform better than solution LIN does in example (c) above. If on the two given paths the utilisation is similar we can expect that a similar amount of sessions will leave both paths. From this it follows that both paths should be given a similar weight although the ratio of the free capacities of the paths is high. Of course, if the free capacities on both paths are high, their ratio will be close to 1, so their weights will be close. This heuristic assumption can be formalised as a criterion.

Let the assigned weight be a continuous, non-negative and monotony increasing function of the free capacity. Weights can be calculated according $w_i = f(f_i)$. The most simple function is $f(x) = x$, the already explained LIN solution.

The mentioned criterion can be formulated against function $f(x)$. For a small difference e, the values of $f(x)$ and $f(x+e)$ should be "close" to each other. We introduce a new function:

$$g_e(x) = \frac{f(x)}{f(x+e)} \tag{3}$$

Since $f(x)$ is monotony increasing, $g(x) \leq 1$ for every $x \in [0;\infty)$, and since $f(x)$ is positive, $g(x)$ is also positive. The criteria can be formulated as

$$g_e(x) > (1 - \epsilon_e). \tag{4}$$

Function $g(x)$ is shown on Fig. 2 for $f(x) = x$, showing that LIN fails to fulfil the requirement at small x values. Though,

$$\lim_{x \to \infty} \frac{x}{x+e} = 1, \tag{5}$$

which means that the linear function fulfils the error ignorance criteria when the network is lightly loaded. The undesired behaviour at small values is correlated to the fact that $f(0) = 0$.

Lemma 1 *If $f(0) = 0$, one can always find an appropriate small x_ϵ value, where $g(x_\epsilon) < (1 - \epsilon)$, which violates the error ignorance criteria.*

Proof If $f(0) = 0$ for any f function, then $g(0) = 0$, and if f is continuous, so is g, too. From the definition of continuance, one can always find an appropriate small x_ϵ value, where $g(x_\epsilon) < (1 - \epsilon)$. ■

We have to define the set of functions that fulfil the error ignorance criteria.

Lemma 2 *If $f(x)$ is continuous, monotony increasing and strictly positive, and if for a given x_0 $f'(x_0) = 0$, then one can find ϵ_e with which $g_e(x_0) = \frac{f(x_0)}{f(x_0+e)} > (1-\epsilon_e)$ follows, satisfying criterion (4).*

Proof From the definitions of the differentiate quotient and the limit it follows that for any e one can find δ_e that satisfies

$$\frac{f(x_0+e)-f(x_0)}{e} < \delta_e. \tag{6}$$

From this,

$$f(x_0+e)-f(x_0) < e\delta_e, \tag{7}$$

$$f(x_0+e)-e\delta_e < f(x_0), \tag{8}$$

and because $f(x_0)$ is strictly positive, it follows that

$$1-\frac{e\delta_e}{f(x_0+e)} < \frac{f(x_0)}{f(x_0+e)}. \tag{9}$$

With $\epsilon_e = \frac{e\delta_e}{f(x_0+e)}$, the lemma is proven. ■

From this it follows that if at zero the function is positive and the derivative is small, we can limit the effect of a small difference e onto the ratio of weights at small free capacities. Therefore, we looked for functions that comply with this condition.

One such function is $e^{\frac{x-a}{b}}$, where a and b are parameters that control how flat the exponential curve is. An interesting feature of this function is that the effect of a difference e is independent of the free capacity:

$$g_e(x) = \frac{f(x)}{f(x+e)} = \frac{e^{\frac{x-a}{b}}}{e^{\frac{x+e-a}{b}}} = \frac{e^{\frac{x-a}{b}}}{e^{\frac{x-a}{b}}e^{\frac{e}{b}}} = \frac{1}{e^{\frac{e}{b}}} = \text{const.} \tag{10}$$

We used this function only up to the point where its gradient is 1, after that we replaced it with the simple linear function because criterion (4) is fulfilled in that domain as shown on Fig. 2. (We will refer this mixed function as EXPLIN.) Fig. 3 shows examples with different parameters.

4 Numerical results

4.1 Test scenarios

Our simple test scenario 1 is based on the topology shown on Fig. 4. Only node a has a load sharing role: it has three alternative next hops towards destination d, the "upper" path of $a-b-f-d$, the "middle" path of $a-c-f-d$ and the "lower" path of $a-e-g-d$. In traffic scenario 1a only node a generated traffic towards destination d. The offered load increased continuously, reaching 100% at 6000 seconds simulation time. This scenario was destined for showing the increasing fluctuations of load splits at higher loads with the simple linear $f(x) = x$ function. In traffic scenario 1b nodes a, f and g generated traffic towards destination d, together filling the available 3000kBps capacity. In order to simulate load shifts in the offered traffic (for which dynamic load sharing should adapt), this 3000kBps total offered load was generated by deterministically modulating the Poisson-arrival processes on the three source nodes. This scenario was meant to show that eliminating oscillation with the proposed functions has the benefit of higher network utilisation. We also investigated a more realistic

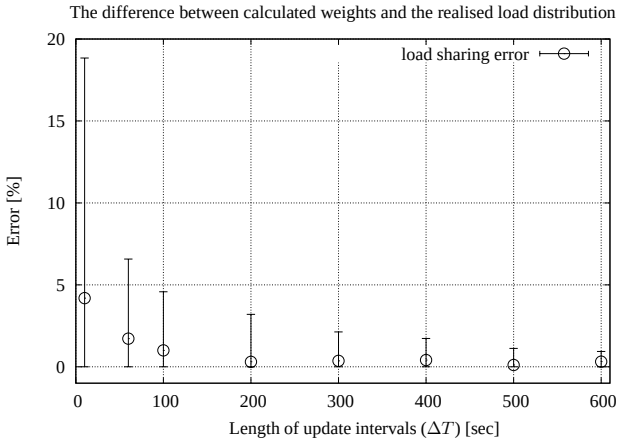

Figure 1. Load sharing errors

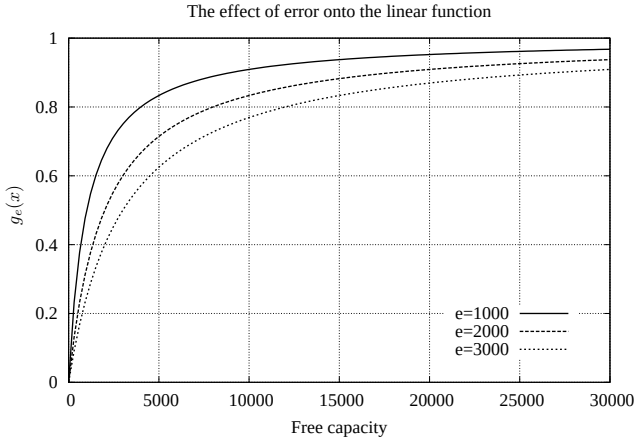

Figure 2. The $g_e(x)$ function for $f(x) = x$

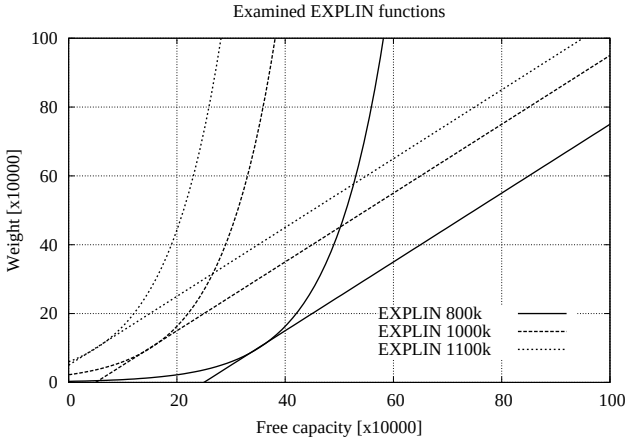

Figure 3. EXPLIN functions

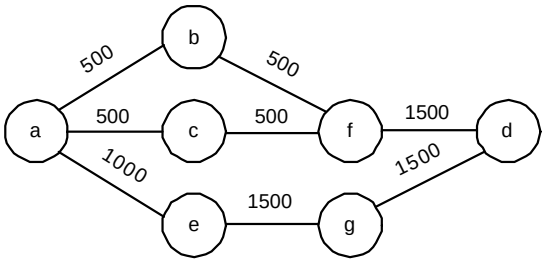

Figure 4. Topology of the test scenario 1.

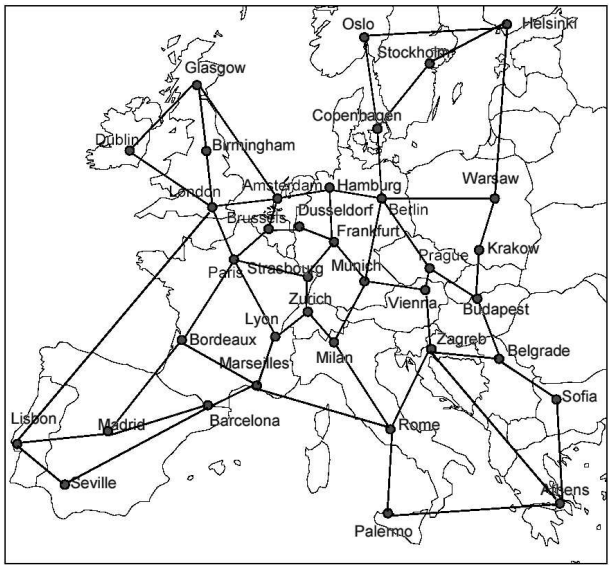

Figure 5. COST-266 “Large” backbone topology

test scenario 2. We used a European backbone topology which was proposed by the European Cooperation in the Field of Scientific and Technical Research (COST) framework [8]. See Fig. 5. A modulated Poisson process, similar to scenario 1b, was used to generate traffic. This test bed is meant to verify that similar results can be observed in a more complex scenario.

4.2 Simulation results

In this section we turn to the evaluation of the weight setup functions. First of all, we would like to show the oscillation problem occurring with the simple linear function when the network is highly loaded. Fig. 6) shows the sharing ratio of the upper and middle, and the lower paths at node a with scenario 1a. As we approach 95% load at 4000

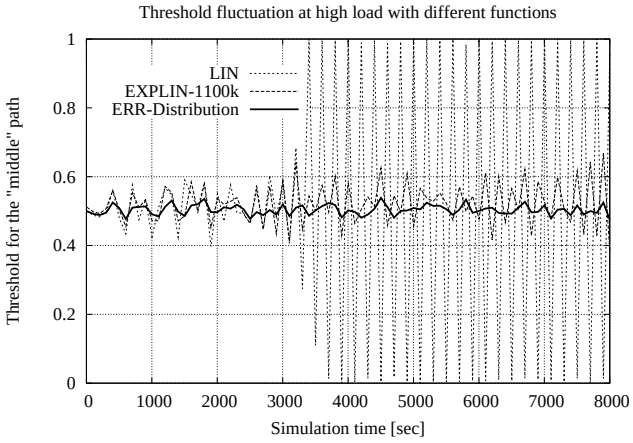

Figure 6. Threshold fluctuation

seconds the LIN function starts severely oscillating. The EXPLIN function effectively decreased fluctuation, and it is also well plotted that the ERR-Distribution method bypasses the problematic of low free capacities, hence no oscillation is observable.

Next, you can see on Fig. 7 the connection blocking results of the different weight assignment procedures with test bed 1b. At small time scales, the linear function performs well. Since it can rapidly contra-react previous over-reactions, this results in the end in low blocking probability. In fact, at smaller time scales the linear solution is second to the ERR-Distribution method. At higher ΔT values, the different capabilities of the methods are clearly observable. The linear function suffers a great drawback at high ΔTs because the weight fluctuation affects more and more traffic. The EXPLIN solutions efficiently reduces the fluctuation. However the graphs suggest that the right (a, b) parameters must be found to achieve good performance where there is a good balance between reduced fluctuation and the capability of reacting to changes. The parameters depend on the length of the update interval (ΔT), the flow arrival and termination distribution and the utilised hash function. The lowest blocking probability could be realized with those solutions that incorporated the expected resource releases into the free capacity. The ERR-Mean method cannot estimate the expected resource releases as precise as the ERR-Distribution, so its performance is inferior to the distribution based variant, but in particular the ERR-Mean method is easier to realise as no knowledge about the distribution function is necessary.

Finally, Fig. 8 shows the results with test scenario 2. Please note that due to heavy processing power needs of the simulation of a large network (8000 seconds required 6 hours on a 2GHz Athlon processor), we could only simulate the LIN, the EXPLIN-1100k and the ERR-Distribution

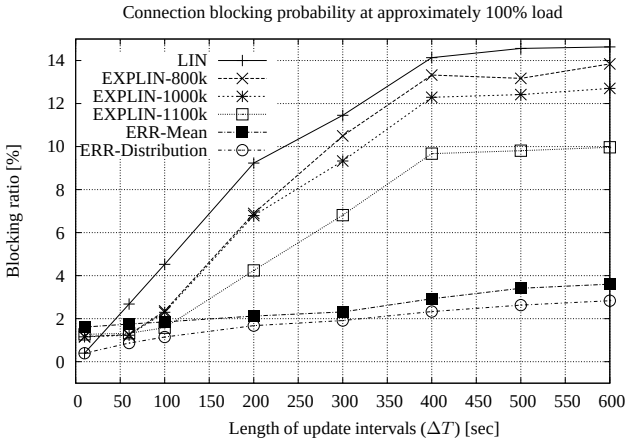

Figure 7. Connection blocking in test scenario 1b.

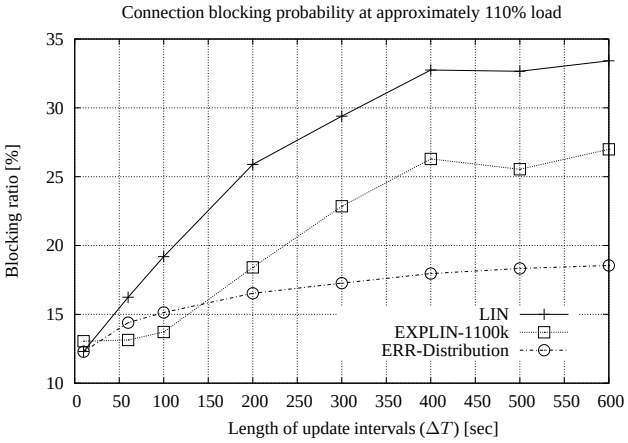

Figure 8. Connection blocking in test scenario 2.

methods. Nevertheless, the outcome of these simulations confirm that similar results can be expected in larger and more complex network scenarios as well.

5 Conclusion

In this paper as the heart of dynamic load sharing architectures we discussed some important aspects of free-capacity based weight calculations. For the investigations we used a novel load sharing framework called CSLLS which is a *thrifty* traffic engineering solution based on a multi-hashing schema. We discovered that it is impractical to set the weights in direct linear proportion of the free capacity because in a highly utilised environment this over-utilises some paths while under-utilises others, resulting in the end in oscillating and instable weights. The key finding in this paper is that for best setup of the load sharing weights one should sum the currently available bandwidth with the bandwidth that is going to be released during the next interval. We proposed two algorithms that estimate the expected resource releases in the next interval based on an estimation of the remaining call holding times of the flows. On the other hand, in practice data about call holding time distribution is rarely available, hence we also suggested a heuristic approach of coping with oscillation and instability, which does not require statistical information. This solution sets the weights according to an appropriate function of the free capacity. We identified some criteria against weight setting functions and also suggested one function complying with the criteria. Our numerical results showed that all these algorithms outperformed the simple linear approach.

References

[1] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao, "Overview and principles of internet traffic engineering," RFC RFC3272, IETF, May 2002.

[2] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," RFC RFC3031, IETF, Network WG, Jan. 2001.

[3] Yufei Wang, Zheng Wang, and Leah Zhang, "Internet traffic engineering without full mesh overlaying," in *Proceedings of IEEE InfoCom 2001*, Anchorage, Alaska, April 2001, IEEE.

[4] Ashwin Sridharan, Roch Guérin, and Christophe Diot, "Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks," Tech. Rep., University of Pennsylvania, 2003, A shorter version appears in INFOCOM'2003.

[5] Curtis Villamizar, "OSPF optimized multipath (OSPF-OMP)," in *Proceedings of the forty-fourth Internet Engineering Task Force*, Minneapolis, MN, USA, Mar. 1999, IETF, vol. 1999. March.

[6] Attila Takács, András Császár, Róbert Szabó, and Tibor Cinkler, "Thrifty traffic engineering through CSLLS," in *Proceedings of the 18th International Teletraffic Congress ITC18*, Berlin, Germany, Sept. 2003, pp. 61–70.

[7] Zhiruo Cao, Zheng Wang, and Ellen Zegura, "Performance of hashing-based schemes for internet load balancing," in *Proceedings of IEEE InfoCom 2000*, Tel Aviv, Israel, March 2000, IEEE.

[8] European Cooperation in the Field of Scientific and Technical Research (COST), "Advanced infrastructure for photonic networks," web page: http://www.ure.cas.cz/dpt240/cost266/index.html.