Liberating TCP: The Free and the Stunts

Jason Valdez
Computer Science Department
Texas State University
San Marcos, TX 78666, USA
jv1150@txstate.edu

Mina Guirguis
Computer Science Department
Texas State University
San Marcos, TX 78666, USA
msg@txstate.edu

Abstract

The performance of a TCP connection is typically dictated by what the network can provide rather than what the application would like to achieve. In particular, the Additive-Increase Multiplicative-Decrease (AIMD) mechanism employed by TCP hinges on its ability to meet specific throughput requirements since it has to respond to congestion signals promptly by decreasing its sending rate. The level and the timing of congestion signals impose strict limitations on the achievable throughput over short time-scales. To that end, this paper presents a new architecture, whereby a set of TCP connections (we refer to them as the Stunts) sacrifice their performance on behalf of another TCP connection (we refer to it as the Free) by picking up a delegated subset of the congestion signals and reacting to them in lieu of the Free connection. This gives the Free connection just enough freedom to meet specific throughput requirements as requested by the application, without affecting the level of congestion in the network. We evaluate our architecture via extensive simulation experiments.

1 Introduction

Motivation: Some classes of applications (e.g., real-time, streaming and gaming) need to aquire/maintian certain guarantees in order to perform adequately. Due to the best-effort nature of the Internet, however, it is very difficult to ensure that these guarantees are met or even to predict what possible guarantees could be provided. Hence, these applications are often left with unspecified guarantees on their quality of service. Research efforts have addressed this problem and proposed two major architectures, IntServ [4] and DiffServ [3]. IntServ architectures require *every* router to maintain per-flow state. Applications make reservations based on their needs. The main problem with IntServ is that it does not scale to a size that is as large as the Internet. Thus, it is limited to small-scale deployments. DiffServ architectures, on the other hand, push traffic management towards the edge of the network, while keeping the core simple. Edge routers maintain and classify flows based on classes. Core routers still need to maintain brief information on how to treat each class. In both architectures, some modifications had to be made to some routers.

The "Free and Stunts" Architecture: In this paper, we propose a new architecture that provides an application with *soft throughput guarantees* over a best-effort network, without *any* modification to routers. Moreover, this is achieved in a completely friendly manner to the network via strictly adhering to the Transmission Control Protocol (TCP) rules. In particular, we envision a set of TCP connections (we refer to them as the Stunts connections) that are willing to sacrifice their own performance on behalf of another TCP connection (we refer to it as the Free connection). This would enable the Free connection to match its throughput to the requested throughput from the application.

TCP employs congestion control mainly via the Additive Increase Multiplicative Decrease (AIMD) mechanism that seeks to constantly probe for available capacity while remaining fair to other TCP flows [5, 9]. Congestion signals (as in dropped/marked packets) signal TCP senders to slow down, by halving their congestion windows. The quantity and the timing of these congestion signals may prevent the Free connection from achieving any throughput guarantees. The main idea behind our architecture is to allow the Free connection to delegate a subset of those congestion signals to the Stunt connections. Thus, the Stunt connections would be the ones that decrease their sending rates instead of the Free connection, which would be liberated (to a larger extent) to match the guarantees requested from the application above. It is important to note that this architecture *does not* impact network congestion in any manner. That's because it ensures that the total decrease in throughput from all the Stunt connections is at least as large as what the Free connection would have decreased, if it were to observe those delegated losses.[1]

[1] It is possible for a single packet loss delegated from the Free connec-

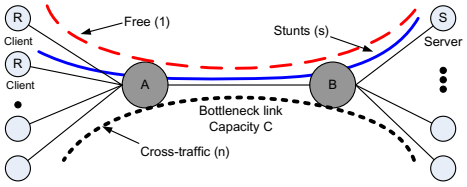

Figure 1. The Free and the Stunts Setup.

Deployment Examples: Internet servers typically serve different forms of media to clients. By employing this architecture, a server can give a particular flow (a media stream) the freedom to match some requested guarantees, while making other flows (bulky file transfers) behave as Stunts. Throughout this paper, we assume the existence of those Stunts (from the server to other clients) and that they can be utilized by our proposed architecture. This architecture can also be used by an ISP to provide differentiated service among its clients, by having some behave as the Free connections, while others play the Stunt roles.[2]

Paper Organization: Section 2 describes our proposed architecture in detail. We present our evaluation in Section 3. Section 4 puts this work in contrast to other related work and we conclude in Section 5.

2 The Architecture

2.1 The Components

We envision a setup composed of a single Free TCP connection and s Stunt TCP connections. Those $s+1$ TCP connections traverse a bottleneck link along with n other TCP connections representing normal cross-traffic. Thus a total of $(1+s+n)$ TCP connections traverse that bottleneck link. Figure 1 depicts this setup. The application running on top of the Free connection requests its throughput requirements through a trace file. This trace file is first checked by a preprocessor for feasibility. If any of the checks fail, another feasible trace is created that is closest to the original trace file; otherwise, the trace file is passed directly to the controller. The controller compares the achievable throughput to the requested throughput over every time instant. Based on the difference, the controller adjusts the ratio of congestion signals to be delegated from the Free connection to the Stunts in order to match the achievable throughput to the requested throughput. A monitor measures the throughput achieved by the Free connection and reports this value back to the controller. Figure 2 represents the different components in the architecture.

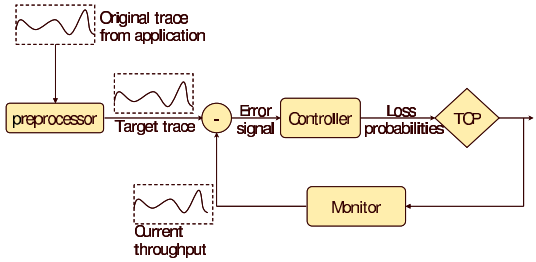

Figure 2. The Architecture.

The trace file: An application specifies its requirements via a trace file which describes the *shape* of the throughput over time. It is composed of two-tuple entries in the form of *time* and *throughput*. An entry of the form $< i, T_i >$ indicates a request of T_i throughput at time instant i.

The preprocessor: The main goal of the preprocessor is to check the feasibility of the trace file and to create another feasible trace, if any of the checks fail. It performs two checks, a slope check and a region check. The slope check ensures that the requested throughput can be attained from one time instant to the next based on the round-trip time (RTT) of the Free connection. For any two successive time instants, i and j, the requested throughput T_j at time instant j, is bounded by:

$$T_j \leq T_i + \frac{j-i}{RTT^2} \quad (1)$$

Since TCP increases its congestion window by 1 packet every RTT, the congestion window at time j, cannot be more than $\frac{j-i}{RTT}$ of the congestion window at time i. Dividing by RTT to obtain the throughput leads to the above equation. The region check prevents the Stunts from achieving zero throughput. Thus for any time instant i, the requested throughput is bounded by:

$$T_j \leq s \times \bar{x}_s \quad (2)$$

where $\bar{x}_s$ is the average expected throughput per Stunt connection. This is a preliminary check and a more strict check is enforced online.

The controller: The controller decides which losses are picked up by the Free connection versus those delegated to the Stunts. The decision is based on the error signal between the current throughput and the requested throughput. We have experimented with an On/Off controller and a Proportional Integral (PI) controller. An On/Off controller, de-

tion to cause more than one Stunt connection to back-off, in order to have the same equivalent effect on the network.

[2]By marking the ECN bit in appropriate packets, an ISP can delegate losses between the Free and the Stunts.

cides the delegation percentage g_i, at time i according to the following equation:

$$g_i = \begin{cases} 0 & x_i > 1.3\bar{3}T_i \\ 1 & otherwise \end{cases} \quad (3)$$

where x_i is the instantaneous throughput of the Free connection at time instant i. An On/Off controller will try to delegate all the losses to the Stunts, whenever the current throughput is not matching the requested throughput. Otherwise, the Free connection will pick up its own losses and react to them. Notice that we compare x_i to $1.3\bar{3}T_i$ as opposed to T_i directly. This is due to the AIMD mechanism and the operation of the controller at short time-scales. A packet loss at $1.3\bar{3}T_i$ will cause the throughput to drop to $0.6\bar{6}$, leading to the target average of T_i, assuming a fixed RTT. The PI controller adjusts the delegation percentage based on the following equation:

$$g_i = g_{i-1} + K \times (x_i - 1.3\bar{3}T_i) \quad (4)$$

where K is a constant the decides the aggressiveness in reaction to the error signal between the current throughput and the requested throughput. The higher the value of K is, the more aggressive the controller would react.

2.2 Delegation of Congestion Signals

The Free and the Stunts architecture capitalizes on the fact that a delegation of a congestion signal from the Free connection to the Stunts will both (1) allow the Free connection to achieve a higher target data rate, and (2) it will not violate *any* TCP congestion control rules.

Since the Free connection can delegate congestion signals, it can continue sending as dictated by the Additive Increase component of the AIMD mechanism, by increasing its congestion window by 1 packet every RTT. Since the requested throughput is slope-checked by the preprocessor, then it should be able to achieve the requested target. However, in some conditions (explained below), the Free connection may not be able to delegate a congestion signal and thus it would have to cut its congestion window in half.

Notice that the impact of a congestion signal is not the same – as far as the network is concerned – whenever it gets delegated, since one connection may have a different congestion window size than the other. Thus, the reduction in the sending rate will not be equivalent.

In our particular case, to make sure that the network sees an equivalent reduction, we first check to see if the total congestion windows (of all the Stunts) is larger than that of the Free connection. If so, then we go through the Stunts, one by one in a round robin fashion, and we halve each one's congestion window, until the total reduction is at least as large as half the Free connection's congestion window size. If not, then we cannot delegate this loss and the Free connection has to react to it in the normal way, since we can only cause each Stunt connection to back-off one time for a given loss. Note that the round robin algorithm may cause the last Stunt connection to decrease its rate by a bit more of what is actually required, since we do not optimize to find the best fit among the Stunt connection's congestion window sizes that would sum to exactly the Free connection's congestion window size. This is wasted bandwidth that is essentially given up by the Stunts to be acquired by all connections. The effect of this is diminished over time as all connections grab more throughput.

The reason we go in round robin fashion is to be fair across the Stunts. TCP fairness in our case is considered globally across the group of Free and the Stunt connections, as they can be abstractly considered as a single entity. The group of Free and Stunts together should not be more aggressive than an equivalent number of ordinary TCP flows.

Paying back the Stunts: In some situations, the Free connection may not need a higher data rate. Either because the requested throughput at some point in time may go under its fair share or its data rate has increased to a point that is above the requested target rate. In both cases the Free connection can easily give up this bandwidth by having the application send less data. However, this slack of bandwidth will be naturally acquired by all connections (Stunts and cross-traffic). We have modified both controllers described above to allow for the reverse loss delegation from the Stunts to the Free connection. Reverse delegation helps the group of Free and Stunt flows to retain bandwidth as a whole, as apposed to releasing it to the network. Furthermore, it allows the Free connection to closely match its target when the target is low (typically below its fair-share). Also, as discussed above, delegation in this case would ensure that the Free connection would have a larger congestion window than the Stunt that is delegating. Otherwise, the Stunt connection cannot delegate a loss and would have to react to it.

Although other behaviors such as timeouts and slow-start can also be delegated, we focus on the AIMD due to the complexity and rareness of those events in comparison to the AIMD.

3 Experimental Evaluation

In [12], we present a nonlinear fluid model for our architecture along with numerical solutions, which we do not present here due to lack of space. We report here on our simulation experiments using NS-2 [6].

The Setup: Figure 1 depicts the topology used where we assume all connections have an infinite supply of data

to transmit. The bottleneck link is configured with RED [7]. The queue size at the bottleneck link is chosen to be $\frac{RTT \times C}{\sqrt{m}}$ as advocated in [1], where C is the bottleneck link capacity and m is the total number of connections traversing the bottleneck. The RED parameter B_{min} is set to 0.25 the size of the queue resulting in the distance between B_{min} and B_{max} being three times B_{min}. Other parameters were chosen to encourage the stability of the average queue size.

Performance Metrics: To measure the effectiveness in matching the achievable throughput to the requested throughput, we propose a weighted variant of the standard "sum-of-squared errors" method. The standard "sum-of squared errors" does not differentiate between the case where the achieved throughput is above the target, versus below the target, since in both cases we may get the same value. So we define the positive variance V^+ to be:

$$V^+ = \frac{\sum(x_i - T_i)^2}{C^+} \quad \forall\ x_i > T_i \tag{5}$$

where C^+ is the number of times (sample points) the achieved throughput is above target. Similarly, we define the negative variance V^- to be:

$$V^- = \frac{\sum(x_i - T_i)^2}{C^-} \quad \forall\ x_i < T_i \tag{6}$$

where C^- is the number of times (sample points) the achieved throughput is below target. To capture the overall performance we use a weighted variance, defined by:

$$V^* = \delta \times \frac{\sum(x_i - T_i)^2}{C^+ + C^-} \tag{7}$$

where δ is a ratio that is given by:

$$\delta = \frac{max(V^+, V^-)}{min(V^+, V^-)} \tag{8}$$

where δ is always greater than or equal to 1. If the matching is achieved ideally, then δ would be 1. A larger value of δ indicates a bias in the matching, either above or below the target, and this would increase the weighted variance in return. The above metrics are computed over an entire simulation experiment.

3.1 Matching the Target Throughput

This set of experiments assess the ability of the Free connection in matching its throughput to different target trace files. Figure 3 shows representative results using three different trace files under different parameters. All results were obtained using a PI controller and with 10 Stunt connections. In each one, we plot the target trace, the throughput of the Free connection and the average throughput across all Stunts. Figure 3 (left) was obtained using a topology with 40 Mbps bottleneck link capacity. In order to provide some variability, 8 of the cross-traffic connections through the bottleneck were randomized at 10-second on/off intervals with the exception of 2 cross-traffic flows which were continuous. This experimentation ensured that the flows through the bottleneck did not experience many timeouts.

Figure 3 (middle and right) were obtained using a topology with 80 Mbps bottleneck link capacity. The number of cross-traffic links was kept constant at 20 connections. Overall, one can see that the Free connection does a fairly good job in matching the target throughput while the throughput achieved by the Stunts changes in tandem. Notice the larger oscillations at higher data rates; these are expected due to the normal behavior of the AIMD mechanism. We see the opposite effect at lower data rates, due to a smaller decrease in bandwidth.

Notice also that the reverse delegation of losses from the Stunts to the Free connection allows the Stunts to achieve higher rates than they would have achieved otherwise.[3] This is evident from Figure 3 (right) from time 150 until around 235 seconds. During this time interval, the Stunts are achieving higher throughput than their fair share, since the Free connection does not need that throughput.

3.2 Impact of the Number of Stunts

To study the impact of the number of Stunt connections on the performance of the Free connection, we vary the number of stunts while holding all other parameters constant and we plot the weighted variance (as given in Equation 7) versus the number of Stunts.[4]

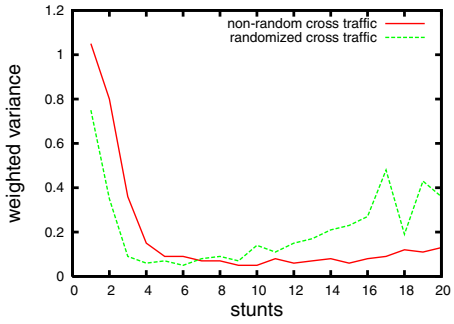

Figure 4. Impact of the number of Stunts.

[3] The slack of bandwidth given up by the Free connection goes directly to the Stunts as opposed to going to the Stunts and the cross-traffic.

[4] As mentioned in Section 1, these Stunt connections exist due to the normal operation of the server. We do not advocate creating them to make this architecture work.

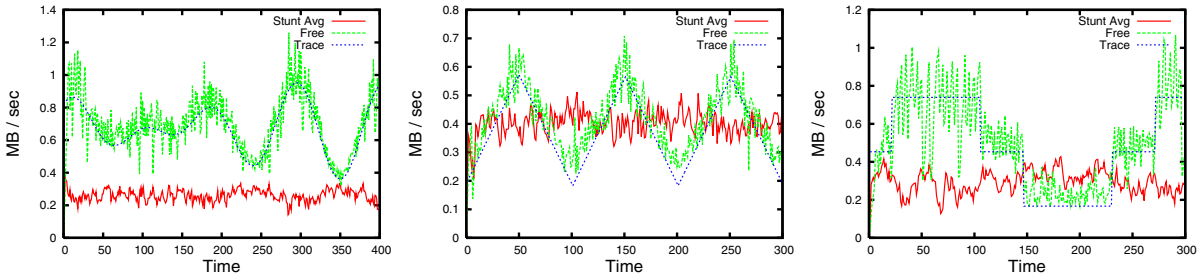

Figure 3. Three simulation traces to assess matching the target throughput.

Figure 4 shows the results obtained (the non-random cross-traffic plot), where each point represent an independent simulation run. One can observe that there is an optimal number of Stunts (around 5 or 6) that minimizes the weighted variance. Increasing the number of Stunts further, not only shows diminishing returns but also harms the performance of the Free connection as indicated by a slight increase in the weighted variance towards the higher number of Stunts.

Figure 5 shows the exact performances with 2, 10 and 20 Stunts, respectively. These plots were generated on topology of an 80 Mbps bottleneck link with 20 cross-traffic connections. Clearly, a very small number of Stunts (2) has a noticeable degrading effect on the performance of the Free connection, which improves with an increased number of Stunts up to a point where is starts decreasing again.

The number of Stunts affect the overall efficiency of the method because they are more than a reservoir of bandwidth for the Free connection. In particular, they adjust the level of congestion in the network for the Free connection to better match the target throughput. If there number is very low, the Free connection would not be able to delegate losses (since we strictly enforce the same reduction in congestion windows from all Stunts). If there number is very large, the network would be more congested and all flows would start to go into timeouts/slow-start which again impact the Free connection in matching the target throughput.[5]

3.3 Impact of Cross-traffic Dynamics

To study the impact of dynamics that arise in practice, we allow a number of the cross-traffic connections to be turned on and off every 10 seconds uniformaly at random.

Figure 6 shows the impact of varying the number of the cross-traffic connections, while keeping the number of Stunts steady at 10. We plot the positive variance, the negative variance, and the weighted variance to better explain a unique behavior observed in this experiment.

[5]One approach to handle this case would be to delegate timeouts, however, our focus in this paper was mainly on the AIMD mechanism as explained earlier.

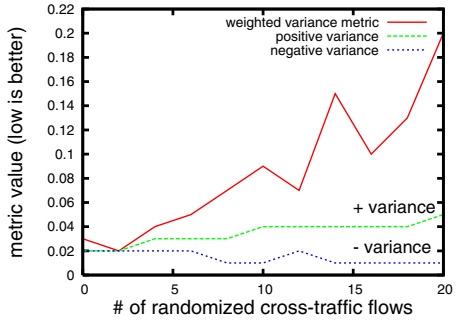

Figure 6. Impact of cross-traffic dynamics

It is clear that the higher the dynamics from the randomized cross-traffic, the harder it is for the Free connection to match the target. However, when we examined the exact performance of the Free connection, we found that its shape was rather intact than deformed, but it was above the requested target. This was confirmed visually in each simulation run and is easy to see by examining the positive and negative variance metrics. Notice how the positive variance grows larger as the negative variance grows smaller. Such divergence increases the weighted variance due to a higher δ. The throughput of the Free connection was above the target because with a larger number of randomized cross-traffic, their combined throughput decreased the utilization at the bottleneck. This caused the Free connection to acquire more than the target because the lower link utilization also resulted in a lower packet loss probability.

Figure 4 (the randomized cross-traffic plot) shows the impact of the number of Stunts when most of the cross-traffic connections (19 out 20) were randomized on a 10-second on/off intervals. The presence of dynamics, coupled by an increase in the Stunts lead to a degraded performance in matching.

4 Related Work

Other than the IntServ [4] and DiffServ [3] architectures and their associated deployment issues, there has been some work that aim to provide soft QoS guarantees through

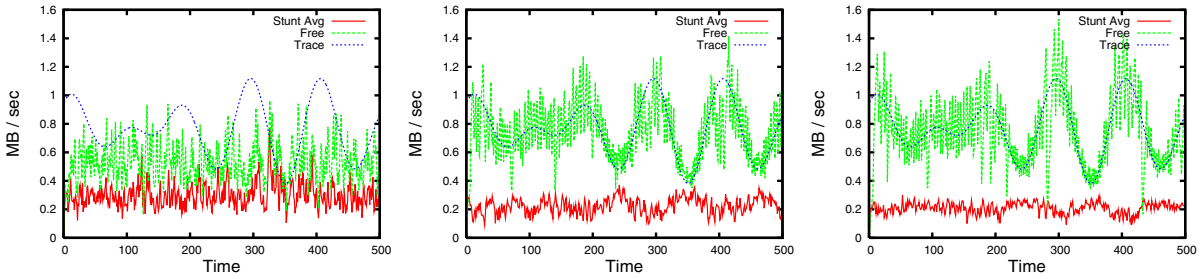

Figure 5. Impact of the number of Stunt connections on the performance. Left plot with 2 Stunts, middle plot with 10 Stunts and right plot with 20 Stunts.

existing technologies as in [2, 8, 10, 11]. The work in [2] focused on managing the end-to-end behavior of TCP connections through sharing congestion information among the connections. The work was focused on applications adapting their sending rates to achieve better performance, rather than meet specific guarantees. In [10] an architecture was proposed that enables applications to express policies, probe and adapt to the observed conditions. This however was done at the end-host, using CPU and memory scheduling. In [8], an elastic tunnel was created using a number of regular TCP connections to provide soft bandwidth guarantees. The cardinality of connections changes in tandem with cross-traffic, to provide soft QoS guarantees. The work only considered constant QoS guarantees. Moreover, it required modifications to edge routes to manage this tunnel. In [11] the authors propose a coordination protocol (CP), which seeks to optimize cluster-to-cluster communication of computing devices across a bottleneck aggregation point. Their proposal entails, among other aspects, giving the flows across the aggregation point the ability to sense the network state and adapt at the end-points.

5 Conclusion

In this paper we have described an end-host architecture that achieves soft QoS guarantees for a TCP connection, which we refer to as the Free connection, based on throughput requirements from the application. This is achieved by delegating some congestion signals to a group of companion TCP connections, which we refer to as the Stunt connections. The architecture strictly adheres to TCP rules and does not impact network congestion in any manner. Moreover, no modifications is required to devices in the communication path. We have shown that the architecture is capable of providing a reasonably accurate targeting between the achieved rate and the target rate with a small number of Stunts. We have assessed the performance of our proposed architecture through new metrics, using numerical solutions and extensive simulation experiments.

References

[1] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing Router Buffers. In *Proceedings of ACM SIGCOMM'04*, Portland, Oregon, August 2004.

[2] H. Balakrishnan, H. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *Proceedings of ACM SIGCOMM'99*, Cambridge, MA, August 1999.

[3] S. Blake, D. Black, M. Carlson, E.Davies, Z.Wang, and W. Weiss. An Architecture for Differentiated Services. *IETF RFC 2475*, December 1998.

[4] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. *RFC 1633*, June 1994.

[5] V. Cerf and L. Kahn. A Protocol for Packet Network Interconnections. *IEEE Transactions on Communications*, 1974.

[6] E. A. et al. UCB/LBNL/VINT Network Simulator - ns (version 2). Available at http://www.isi.edu/nsnam/ns/.

[7] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *Transactions on Networking*, 1(4):397–413, August 1993.

[8] M. Guirguis, A. Bestavros, I. Matta, N. Riga, G. Diamant, and Y. Zhang. Providing Soft Bandwidth Guarantees Using Elastic TCP-based Tunnels. In *In proceedings of the 9th IEEE Symposium on Computer and Communications (ISCC'2004)*, Alexandria, Egypt, July 2004.

[9] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of ACM SIGCOMM'98*, Stanford, CA, August 1988.

[10] G. Molenkamp, M. Katchabaw, H. Lutfiyya, and M. Bauer. Managing Soft QoS Requirements in Distributed Systems. In *Proceedings of ICPP Workshop*, Toronto, Canada, August 2000.

[11] D. Ott and K. Mayer-Patel. An Open Architecture for Transport-level Protocol Coordination in Distributed Multimedia Applications. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, 3(3), August 2007.

[12] J. Valdez and M. Guirguis. Liberating TCP: The Free and the Stunts. *Computer Science Department, Texas State University - San Marcos. Technical Report*, January 2008.