An Interval Approach to Congestion Control in Computer Networks

Scott Starks

***Abstract*— It is important to dynamically tune communication network parameters in order to improve the performance of the network and to prevent congestion. All parameters can be tuned, including the size of a packet, the retransmission timeout (i.e., the amount of time to wait for an acknowledgment), etc. Tuning is usually done by applying some predefined linear transformation to the current values of these parameters: e.g., add a constant, subtract a constant or multiply the current value by a constant. If we fix the parameters of these transformations once and forever, we still have some congestion problems.**

As an attempt to avoid this difficulty, Van Jacobson proposed to use nonlinear transformations (namely, a specific family of fractionally linear ones). His simulation results showed that this new tuning algorithm really worked well. However, the resulting behavior is still sometimes highly unstable.

In [6, 10], a linear congestion control with adaptive coefficients has been proposed. This tuning method turned out to perform much better that the non-adaptive non-linear one.

In this paper, we use interval approach to explain why linear methods perform better than non-linear ones.

I. Introduction

A. Why do we need to tune?

It is important to dynamically tune communication network parameters in order to improve the performance of the network and to prevent congestion [3].

Congestion can occur for every network that has routers and/or gateways that connect several smaller networks because the resources of these routers are limited and therefore they form a bottleneck for the overall network. A typical example of such a network is the Internet (the same problems occur, however, for every Wide Area Network (*WAN*)). Namely, the observations show the following pattern: the number of messages varies from time to time and inevitably a moment comes when it exceeds the capacity of a router's or gateway's queue. Then some messages are simply dropped from the network. So the computer that sent the dropped message waits for the message acknowledgment (*ack*) for some time (*retransmission timeout*), and when this time expires and there is no ack, it sends the same message again. As a result the same messages can go through the network several times. Therefore the number of the messages that are still in the network increases rapidly: because new messages come and the old messages are retransmitted again and again. As a result the queues become longer, more and more messages are dropped and we are on the edge of congestion collapse.

Congestion problems can become more serious in the case of a *distributed file system (DFS)*, i.e., when we try to access files remotely. The reason why it is more difficult for a network to deal with distributed file system is evident: in addition to the usual messages it now has to move files, and those files can be long, so the total number of messages in the network increases. Some network protocols take care of that problem and so adding a distributed file system in the majority of cases does not worsen the congestion problems; an example of such a protocol is the Transmission Control Protocol (TCP) [1]. However, the most widely used distributed file system is the Network File System (NFS), and the majority of its implementations use another protocol: the User Datagram Protocol (UDP). The reason why the designers preferred UDP is that UDP up to now has been better in performance (faster, etc) than TCP. The users still use NFS over UDP because they have practically no choice: NFS over TCP is not yet widely available; and even if it was, its performance is still worse [5]. NFS in general, and NFS over UDP in particular, works fine with Local Area Networks (LAN). In principle it is possible to use NFS with WANs as well. However, the researchers are extremely cautious about that. Experiments with the existing implementations show

The author is with Department of Electrical Engineering, The University of Texas at El Paso, El Paso, TX 79968.

that as the number of users increases the behavior of the network deteriorates [2, ?]. So they conclude that NFS over UDP might lead to congestion collapse if it were widely used (see, e.g., a footnote on p. 321 of [1]).

In view of all that we really have to dynamically change the parameters of the services that use the network in order to avoid or overcome congestion. So whenever we encounter congestion (or by some means know that we are close to it) we change ("tune") the parameters. So the problem is: how to tune so that congestion is avoided?

B. What to tune?

We want to enumerate the retransmission parameters of the network. Some of the network protocols allow the user to change these parameters, in some of them the user has no access to them, and the method of changing (tuning) them must be incorporated into the implementation of this protocol by the designer of that implementation. Let's enumerate the main parameters and briefly explain what they mean:

- *Window size* (usually denoted by W). A window size is the maximum number of packets that a user is allowed to send before getting an acknowledgment: for example, if $W = 10$ then after sending 10 packets he must wait until some ack comes before he can send the 11th packet. Some protocols do not have such a limitation (e.g., UDP). The protocols for which W is defined are called *window-based* protocols. An example of such a protocol is TCP. The user never has any control over the window size, it is determined by the implementation.
- *Interpacket interval* (usually denoted by I). It is the time interval between packets. This parameter corresponds to a congestion-avoiding method that is an alternative to limiting the window size: namely, after the user sends a packet he cannot send the next one until after some time I elapses.
- *Retransmission Timeout* or *timeout* for short (it is usually denoted by τ_{out}). A timeout is the amount of time during which the node waits for an acknowledgment. If the ack does not come after τ_{out} seconds then the node sends the same packet once again. In TCP, this parameter is controlled by the implementation, so the only way to change it is to incorporate the changes into the implementation. In UDP the user can choose their own τ_{out}.
- *Packet size* (usually denoted by P). It is the size of the packet. There can actually be several packet sizes for the different stages of the communication process. First the information is read or written in quanta that are called the *application-level packet size*. Then depending on the transport protocol these quanta are combined (or subdivided, if necessary) into packets of some other packet size. And, finally, before the signals go into the real physical network they must again be subdivided into packets whose size is not bigger than the maximum physical packet size. The user can control the application-level packet size (for example, NFS packet size), all other packet sizes are bounded by the protocol(s) and the physical network used.

C. How are these parameters tuned now?

Before we present the formulas let's choose the notations. The new (current) value of a network parameter is usually marked by a subscript c (c stands for current) and the previous value by p (p stands for previous). For example, the current value of the window size is denoted by W_c and the previous value by W_p.

The window size is usually changed as follows [1, 9]: if the network becomes congested (we can tell it by the fact that one of the packets is lost) then we must decrease the window size. If, vice versa, no congestion signal appears, that possibly means that there is some additional room in the network, so we can increase W a little bit. For decrease the new value W_c equals to dW_p (this formula is called *multiplicative decrease*). Here d is a positive constant that is smaller than 1. In [1] $d = 0.5$ is proposed, but the authors of [9] claim that $d = 0.875$ is better. Another possibility is *additive decrease* $W_c = W_p - d$, where d is some positive constant, but experimentally it proves to be worse [9]. For increase usually an additive algorithm is used: $W_c = W_p + c$.

The changes in the interpacket interval are usually also described by linear formulas: for increase $I_c = dI_p$ with $d > 1$, and for decrease $I_c = I_p - d$. However, these algorithms sometimes lead to bad oscillations [4, 8, 1]. Therefore in [1] a different update algorithm is proposed for decrease: $I_c = \alpha I_p/(\alpha + I_p)$, where α is some positive constant. According to his experimental data, this algorithm appears to perform well.

Tuning the timeout usually requires more complicated algorithms that take into consideration not only the previous value of the timeout and whether there is a congestion or not, but also the whole history of packet transmissions. Namely, these algorithms are based on the values of round-trip times t_i for all the previous packets, i.e., the times from the moment when the packet was sent to the moment when an acknowledgment was received by the sender. From these data we can estimate an average $a = (\sum t_i)/n$ and a mean square deviation $\sigma = (\sum(t_i - a)^2/(n-1))^{1/2}$. Then the timeout so that practically all known round trip times t_i are smaller than τ_{out}. Following the standard statistical rules, we set $\tau_{out} = a + k\sigma$, where $k = 2..4$. In [1] $k = 2$ is recommended for TCP; the experiments of [5] show that for UDP $k = 4$ is a better choice.

Proposed methods of tuning the packet size are just the same as for windows: for example, for NFS read and write sizes Nowicki [7] proposed to use multiplicative decrease and additive increase.

D. Non-linear and adaptive linear tuning

As we have seen, the choice of the tuning procedure can be crucial. Right now the main tuning procedures are linear ones, in which the new value x_c of the tuned parameter x is a linear function of the previous value: $x_c = ax_p + b$ for some constants a and b. These linear tuning procedures work more or less fine.

The first attempt to try a non-linear procedure (fractionally linear, by Jacobson) immediately lead to the radical performance improvement.

In [6, 10], the intelligent control methodology was used to transform the experience of expert system administrators into a linear congestion control with adaptive coefficients. This control turned out to perform much better that the proposed non-linear one.

This is an empirical result. We did not compare our adaptive linear control with arbitrary non-linear ones. So, before we recommend to use the adaptive linear congestion control, it would be nice to analyze this problem theoretically, and either prove that linear control is better, or find a non-linear control that is better than the linear one.

This is the problem that we deal with in the present paper.

E. Why is this problem difficult to solve?

These problems are difficult to solve because the network is a very complicated system. There is no known way to estimate its performance under different tuning rules other than performing a time-consuming simulation of the whole network. Therefore we cannot really compare a lot of different tuning functions, and even so if we experimentally choose one of them, it will still be possible that we missed the best tuning algorithm.

II. Main Idea

Our main idea is to use *intervals* instead of numbers.

- We will first explain this idea on the example of an interpacket interval. If we fix the interval I, this means that whenever we send a packet at some moment of time t_0, then and the next packet appears during an interval $[t_0, t_0 + I]$, then we do not send this packet. In essence, the number that is called an interpacket interval is actually a characteristics of the actual *interval*: in this example, an interval $[0, I]$.
- Similarly, a timeout τ_{out} is a characteristic of the time *interval* during which we wait for the ack.
- The window size W means that the actual number of unacknowledged messages initiated by this node can be any number from the interval $[0, W]$.
- etc.

What did we gain by rephrasing the numerical characteristics in terms of intervals? At first glance, we only made the description more complicated:

- instead of a single number, we must now describe the interval, i.e., two numbers (its endpoints);
- the tuning transformation f, that was previously described as a function from numbers to numbers, must now be described as a function from intervals to intervals (i.e., as two functions of two real variables).

However, we did gain something: namely, now, we can apply reasonable invariance demands to describe the tuning function f from intervals to intervals (similar invariance demands are used, for a different purpose, in [11]).

The first invariance condition is *scale-invariance*. We want to design a function f that transforms the initial interval $\mathbf{a} = [a^-, a^+]$ into the tuned one $f(\mathbf{a})$. In order to describe the physical quantity $a^\pm$ (e.g., time) in numerical terms, we must fix a unit. It is natural to demand that the transformation f must not depend on what exactly units we have chosen: be it a nanosecond, a millisecond, or something else.

If, instead of the original unit (e.g., sec.), we use a new unit that is λ times smaller (e.g., millisec.), then the numerical values will increase λ times, and instead of $\mathbf{a} = [a^-, a^+]$, we will have

$$\lambda \cdot \mathbf{a} = [\lambda \cdot a^-, \lambda \cdot a^+].$$

So:

- If we use the original unit, then the initial interval domain is described as $\mathbf{a} = [a^-, a^+]$. If we apply the function f to this interval, we get $f(\mathbf{a})$.
- If we use a unit that is λ times smaller than the original one, then the same initial interval will be described by new numerical values:

$\lambda \cdot \mathbf{a} = [\lambda \cdot a^-, \lambda \cdot a^+]$. If we apply the same transformation to these new values, we get the interval $f(\lambda \cdot \mathbf{a})$.

We want these two intervals to coincide:

- the interval $f(\mathbf{a})$ in the old units, and
- the interval $f(\lambda \cdot \mathbf{a})$ in the new units.

To compare these intervals, we must express both intervals to one and the same unit. For example, if we transform $f(\mathbf{a})$ to new units, we will get $\lambda \cdot f(\mathbf{a})$. This new interval must coincide with $f(\lambda \cdot \mathbf{a})$:

$$f(\lambda \cdot \mathbf{a}) = \lambda \cdot f(\mathbf{a}). \quad (1)$$

A similar condition can be formulated if formalize the idea that the transformation f should not depend on the choice of the starting point for time (i.e., if we choose a new starting point that is τ time units before the original one, then the numerical expression for the interval would change from $[t_0, t_0 + I]$ to $[t_0 + \tau, t_0 + \tau + I]$). This invariance can be expressed as:

$$f(\mathbf{a} + a) = f(\mathbf{a}) + a. \quad (2)$$

Now, we are ready for the formal definition:

III. Definition and the Main Results

Definition.

- We say that a function f from intervals to intervals is *scale-invariant* if it satisfies condition (1) for every interval $\mathbf{a}$ and for every positive real number $\lambda > 0$.
- We say that a function f from intervals to intervals is *translation-invariant* if it satisfies condition (2) for every interval $\mathbf{a}$ and for every real number a.

THEOREM. [11] *Every translation-invariant and scale-invariant function from intervals to intervals has the form*

$$f([a^-, a^+]) =$$
$$[\frac{a^- + a^+}{2} + \frac{a^+ - a^-}{2} \cdot b^-, \frac{a^- + a^+}{2} + \frac{a^+ - a^-}{2} \cdot b^+]$$

for some real numbers $b^\pm$.

According to this result, the endpoints of the transformed interval linearly depend on the endpoints of the initial interval. In other words, *reasonable invariance demands lead to linear tuning formulas.* This result explains why linear tuning formulas turned out to be empirically the best.

References

[1] Van Jacobson, "Congestion avoidance and control", *Proceedings of the SIGCOMM 88 Symposium on Communication Architectures and Protocols*, Stanford CA, 1988, pp. 314–329.

[2] C. A. Kent and J. C. Mogul, "Fragmentation considered harmful", In: *Proceedings of the SIGCOMM 87 Workshop on Frontiers in Communications Technology*, Stowe, VT, August 1987, pp. 390–401.

[3] L. Kleinrock, "On flow control in computer networks", In: *Proceedings of the International Conference on Communications*, June 1978, pp. 27.2.1–27.2.5.

[4] C. Kline, "Supercomputers on the Internet: a case study", In: *Proceedings of the SIGCOMM 87 Workshop on Frontiers in Communications Technology*, Stowe, VT, August 1987.

[5] R. Macklem, "Lessons learned tuning the 4.3BSD Reno implementation of the NFS protocol", *Proceedings of the USENIX Winter Conference*, Dallas, TX, 1991, pp. 53–64.

[6] P. Narasimhamurthy, **Application of intelligent control to congestion in computer networks**, Master Thesis, Department of Electrical Engineering, University of Texas at El Paso, December 1992.

[7] B. Nowicki, "Transport issues in the network file system", *Computer Communication Review*, March 1989, pp. 16–20.

[8] W. Prue and J. Postel, **Something a host could do with source quench.** ARPANET Working group request for comment, DDN Network Information Center, SRI International, Menlo Park, CA, July 1987, RFC–1016.

[9] K. K. Ramakrishnan and R. Jain, "A binary feedback scheme for congestion avoidance in computer networks", *ACM Transaction on Computer Systems*, 1990, Vol. 8, No. 2, pp. 158–181.

[10] S. Starks, V. Kreinovich, and P. Narasimhamurthy, "How to avoid congestion in computer networks", In: L. Hall, H. Ying, R. Langari, and J. Yen (eds.), *NAFIPS/IFIS/NASA'94, Proceedings of the First International Joint Conference of The North American Fuzzy Information Processing Society Biannual Conference, The Industrial Fuzzy Control and Intelligent Systems Conference, and The NASA Joint Technology Workshop on Neural Networks and Fuzzy Logic, San Antonio, December 18–21, 1994*, IEEE, Piscataway, NJ, pp. 466–469.

[11] K. C. Wu, "Interval methods in mobile robot control", *These Proceedings*, 1994.