A case for queue-to-queue, back-pressure-based congestion control for grid networks

Marc Herbert and Pascale Vicat-Blanc/Primet
ENS-Lyon, 46 Allée d'Italie
69364 LYON Cedex 07
tel : +33 4 72 72 80 00, fax : +33 4 72 72 80 80
Email: Marc.Herbert@free.fr

***Abstract*— Standard "new-Reno" TCP faces some performance limitations in very high throughput IP WAN networks, (*e.g.*, computing grids) due to a long end-to-end congestion feedback loop and a conservative behaviour with respect to congestion. We investigate breathing new life into the following idea: implement aggressive congestion control via hop-by-hop, link-level flow control (back-pressure). Studying TCP's behaviour at the packet level on an emulated WAN link allows us to explain: a) the dependence between TCP performance and burstiness at the Ethernet level; b) the performance and fairness gains obtained by the activation of Ethernet 802.3x flow control. Recognising that data moves not from hop to hop but from queue-to-queue, regardless of the layer, we propose to refine previous hop-by-hop approaches to a new "Network of Queues" (NOQ) approach, where each network buffer protects itself against overflow, thus preventing global network congestion. We finally identify advantages and limitations and estimate implementation costs of this approach.**

Keywords: Wide Area Networks, flow control, back-pressure, transport, performance

I. INTRODUCTION

In packet networking, congestion events are the natural counterpart of the flexibility to interconnect mismatched elements and freely multiplex flows. Managing congestion in packet networks is a very complex issue. This is especially true in IP networking where the priority was on freely interconnecting an incredible number of networks of very different kinds [1]; it would be unrealistic to require every Internet node in the world to implement a common sophisticated congestion scheme, independently from its peculiar technology. As such, the only congestion signal received by TCP from IP is usually packet loss, or exceptionally ECN [2]. Faced by this (lack of) information, TCP adopts a careful congestion avoidance behaviour [3], with complex dynamics. Since TCP's congestion control is implemented end-to-end, the delay between feedback and control is the round trip time, and its dynamics become "slow" on wide area networks. Adding issues like fairness with legacy flows, TCP has the issue of poor resource utilization [4] and does not fit the needs of biologists or physicists wanting to transfer gigabytes of data routinely on computing grids.

Today's proposed enhancements to TCP tackle this problem in different ways, while trying to retain maximum backwards compatibility with legacy implementations. Highspeed TCP [5] and Scalable TCP [6] increase the aggressiveness in high-performance contexts while trying to stay fair to standard TCP flows in legacy contexts. TCP Vegas [7] and FAST [8] manage IP congestion optimally by leveraging other available congestion information (Round-Trip Time variations, Explicit Congestion Notification, etc.) to regulate rate of transmission, aiming at a fine control of buffer filling in routers. The promising XCP proposition [9] departs further from today's standards, implying a costlier migration path: it proposes a new cooperative congestion control scheme featuring a precise congestion window indication going back from routers to senders.

This paper[1] starts from a more experimental approach: in section II, we first analyse detailed flow traces in order to understand (at the packet level) congestion and performance dependence on link peculiarities. Then the use of Ethernet's flow control protocol (IEEE 802.3x [10], [11]) demonstrates fair, stabilized and wire rated performance. Section III explains why congestion control cannot be well implemented purely layer-wise, ignoring interaction between layers. In section IV we describe our *Network of Queues* (NOQ) proposal for managing network congestion using generalized back-pressure.

II. EXPERIMENTS

A. Platform

In order to gain insights on congestion and flow control at the packet level in a real-world environment, and to understand how this finally relates to TCP *throughput* and *fairness*, we designed the WAN emulation platform pictured in figure 1. A couple of senders are injecting TCP flows into an Ethernet switch EdgeIron 10GC2F (Foundry Networks), using Gigabit Ethernet links (large black lines). The bottom right part of the figure emulates a 100 Mb/s Wide Area Network link, with a 12 ms latency (one-way). Packet captures are performed at the input of this link. The latency is emulated thanks to an (over-provisioned) NISTNet [12] box, configured with a 12 ms one-way delay. All machines run a Debian 2.4.20 Linux kernel. Socket buffer size on senders and receiver were set high enough (1 Mo), so that TCP's end to end flow control (receive

[1]This second, electronic version is slightly revised compared to the 1st version printed in the proceedings. In particular the end of section II-D is updated.

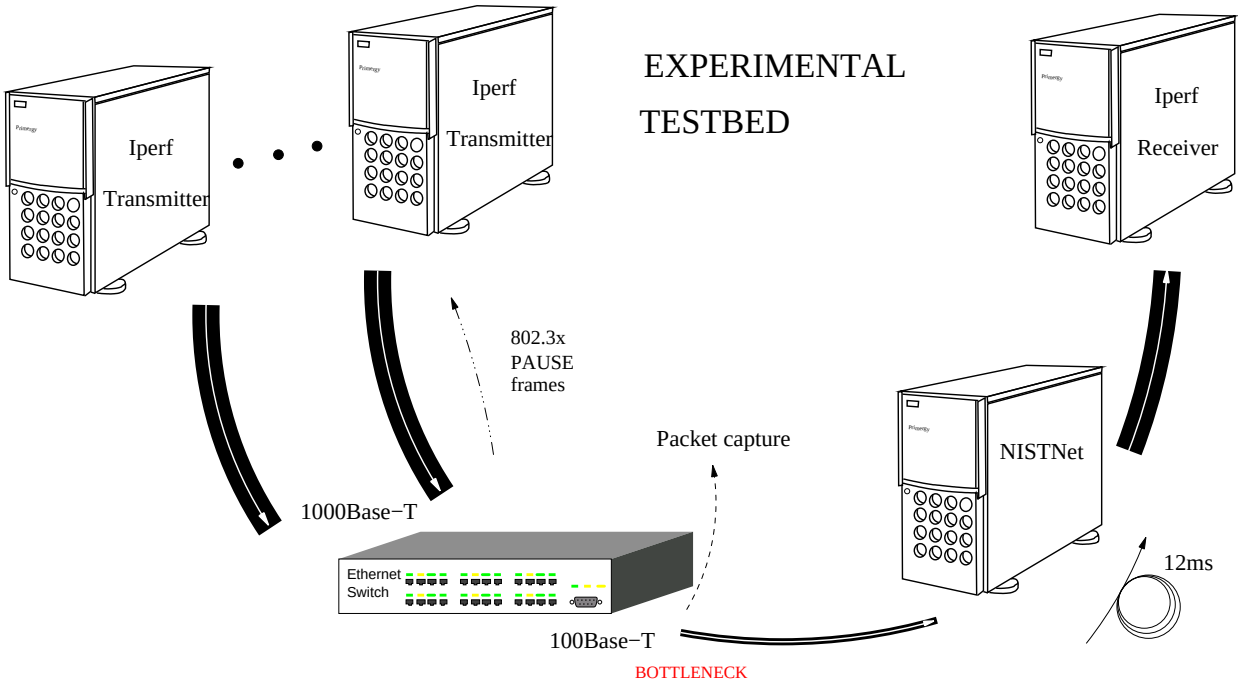

Fig. 1. Test-bed

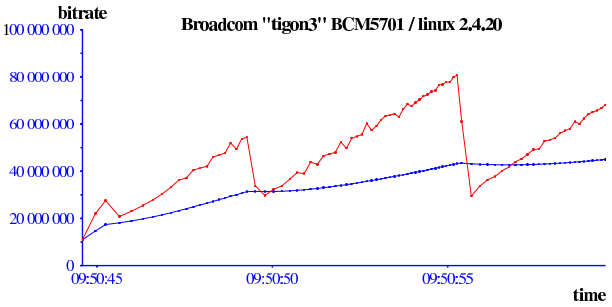

Fig. 2. Usual vanilla-TCP performance

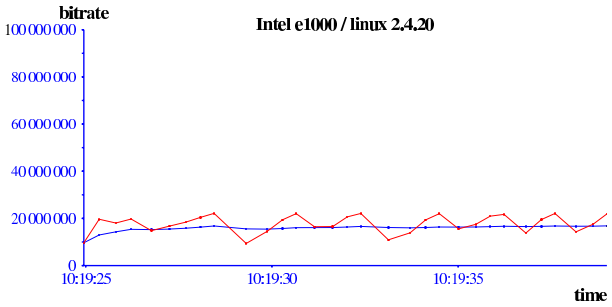

Fig. 3. Ethernet bursts harm TCP

window) is not an issue here. The emulated WAN link has only 100 Mb/s capacity in order to systematically generate congestion.

B. Throughput

The first experiments are the simplest ones (*without* Ethernet 802.3x flow control), in order to have reference values as close as possible to well-known, real cases, and to validate the experimental platform. At first only one of the Gigabit Ethernet senders, a Broadcom "tigon3" BCM5701 Interface, is used to send a TCP flow during 15 s through the 100 Mb/s bottleneck, in order to create congestion inside the switch. Figure 2 shows the throughput, averaged with a 700-packet window, as a function of time. The result is the well-known TCP "sawtooth" oscillation. The average throughput is about 50 Mb/s. As expected, sudden performance drops correspond to packet drops due to buffer overflows in the Ethernet switch: the congestion window is halved at these points [3]. The other and smoother curve on the same figure gives the average throughput *since the start* of the connection.

Link's burstiness considered harmful: Figure 3 shows the throughput after changing *only* the Ethernet NIC/driver: Intel 82543GC/e1000 instead of Broadcom BCM5701/bcm5700. Surprisingly, the performance oscillates around 20 Mb/s. A deeper analysis of packets traces revealed that the default NIC and driver settings seem to often aggregate outgoing packets (as an unfortunate CPU performance optimization), frequently overflowing the limited buffering of the switch. The reason for the better performance of the bcm5700 interface and driver (Fig. 2), is that they run by default an *adaptative* coalescing scheme that prevents interrupt batching at low packet rates (in order to preserve latency).

Another interesting consequence of these bursts is that the round-trip time has a corresponding bursty behaviour. For instance, the number of ACKs deviating noticeably above the nominal RTT value (more than 1 ms late) is much larger than

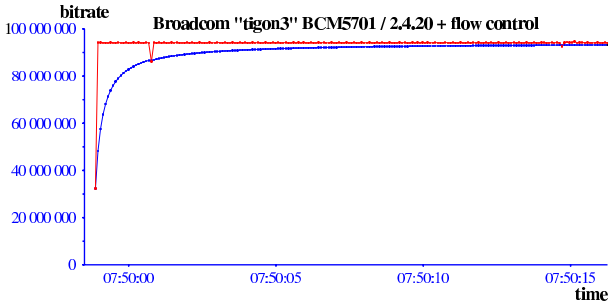

Fig. 4. BCM5700 + 802.3x

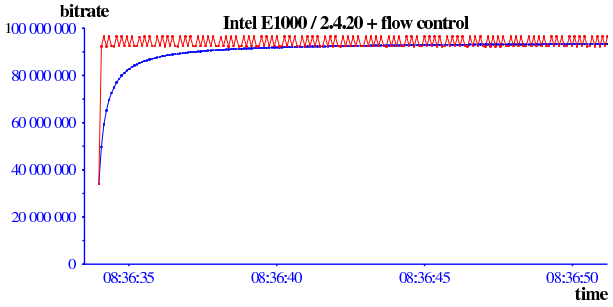

Fig. 5. E1000 + 802.3x

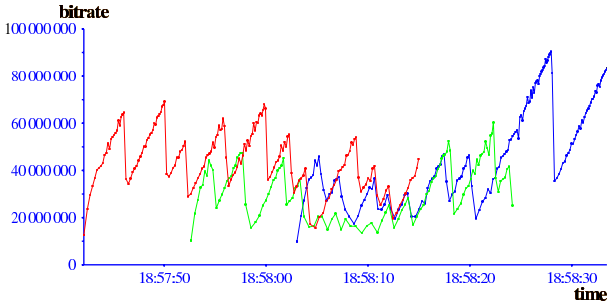

Fig. 6. 3 hosts compete for the bottleneck

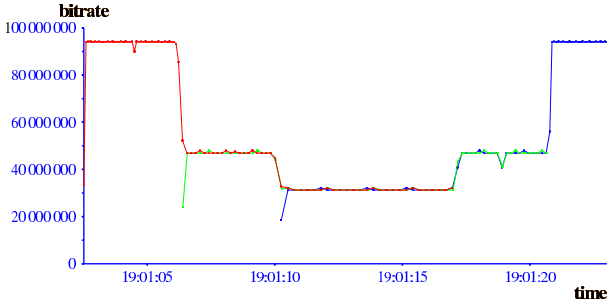

Fig. 7. 3 competing hosts + 802.3x

that for the bcm5700: 3% compared to only 0.1%. Since some of the alternatives to TCP currently proposed (see section I) base their estimations of network congestion upon *round trip time* variations, the local, link-level, high-frequency jitter induced by this "mis-shaping" will add noise to their estimates, in addition to existing issues like undersampling [13]. Time averaging could reduce this noise, but would further slow responsiveness.

802.3x, wire-speed performance and robustness: In figure 4, we come back to the bcm5700 interface, but now activate the IEEE 802.3x flow control protocol. After a very short ramping up phase (TCP's slow start), this has the effect of bringing the throughput up to the wire rate of the emulated WAN bottleneck.

Examination of the packet trace shows that the small and transient throughput drop in the first seconds is due to a lost packet, which triggered TCP's congestion control (uselessly in this experiment). This packet loss may have come from a transmission error.

Figure 5 combines 802.3x and the bursty Intel interface. The burstiness of this interface is clearly visible on the graph, despite the 700-packets averaging window used for the plot. Closer packet trace inspection shows bursts of different scales, the most evident one being bursts of approximately 35 packets every 4 ms. However, thanks to 802.3x back-pressure, packet loss is prevented and the full 100 Mb/s wire rated is still obtained, (compared to only 20 Mb/s without 802.3x). Back-pressure effectively protects queues in the switch against congestion, decoupling Ethernet's burstiness from application performance.

C. Fairness

Three hosts now compete for the same 100 Mb/s bottleneck. They start to transmit one after another, staggered by 10 s. Each connection is 30 s long. Again, we start with standard TCP/IP *without* 802.3x: on figure 6, we can see that the cooperative AIMD algorithm of TCP actually gives a approximately fair share of the capacity to each flow. Activating Ethernet PAUSE frames towards the three senders (figure 7) is quite interesting. Packet losses are prevented. Whenever a new flow appears or disappears, each flow gets the exact fraction of the total 100 Mb/s available almost instantly ("almost" because of TCP's slow start). The aggregate throughput is always equal to the capacity of the bottleneck, and the switch's implementation of 802.3x achieves a perfectly fair sharing between the incoming flows.

The final experiment (figure 8) shows three flows competing *inside the same back-pressured* Linux sender. The aggregate throughput is still wire-rate. However, contrary to the Ethernet

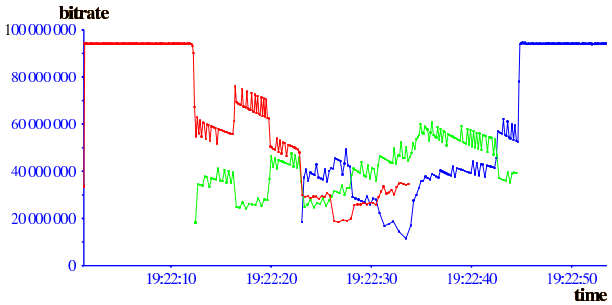

Fig. 8. 3 flows competing *inside* a host + 802.3x

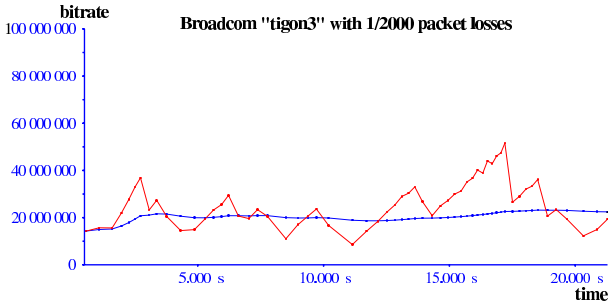

Fig. 9. Performance impact of losses

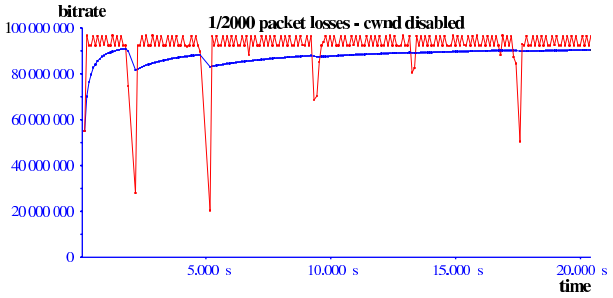

Fig. 10. Disabling TCP congestion control

switch, the Linux kernel does not *fairly* share the bottleneck between sockets, but *randomly* depending on kernel scheduling issues. Investigating Linux's TCP kernel code shows that congestion in queues *inside* the host is *not* handled like network congestion, but rather like some transient hardware error instead, and the congestion window is thus not decreased.

D. Disabling TCP congestion control

Since the network bottleneck of our experiment is able to directly regulate the sender thanks to Ethernet flow control, TCP congestion control (*cwnd*) is not needed anymore. It can even be harmful as previously seen in figure 4, since it conservatively considers all packet losses as congestion signs. Accordingly, this last experiment demonstrates the benefit of disabling TCP's congestion control when it is not needed. Our modifications to the Linux kernel 2.4 allow disabling TCP's congestion control on an per-interface basis. The user can force this disabling or ask for a "safer" mode that checks the state of 802.3x flow control on the Ethernet link before acting.

In addition to emulating latency, we also used NISTNet to generate random packet drops. First, figure 9 shows the performance impact of a high random packet loss rate (1 out of 2000 packets in average) on the vanilla TCP flow of the very first experiment (figure 2). Next, figure 10 shows the performance benefits achieved by "trading" TCP's congestion control for Ethernet flow control: despite the same high packet loss rate, performance is close to the wire-speed of the bottleneck (which is ten times smaller that the sender's link).

However, figure 10 shows that the application bitrate is still not perfectly wire rate: (1) performance takes some time to ramp up (about 100 ms) and (2) some short but deep performance drops also happen later.

A closer inspection of the packet trace shows that the ramp up time to wire rate (4-RTTs) is due to a Linux peculiarity: a "receiver slow-start" algorithm. The receive buffer is not fully advertised at the start of the connection but progressively increased instead. We could not really understand the motivation for this, but could fix it easily in Linux kernel code.

As for the subsequent performance drops, we traced them to a big default transmission queue size (1000 packets) inside the ethernet driver (below TCP). This queue fills up and creates an internal 160 ms long latency inside the sender, in addition to the 12 ms emulated link latency, artificially and considerably increasing the $throughput \times delay$ product. Since SACKs do *not* guarantee any reception but may be legally reneged, a couple of drops are then enough to quickly exhaust the TCP's socket buffer on the sender. A huge socket buffer size on the sender or a smaller driver transmission queue are enough to fix this issue.

III. From hop-by-hop, to queue-to-queue

The case against a layered *congestion control approach:* one may question the necessity of using a low-level flow control technique such as IEEE 802.3x to implement back-pressure: why not simply manage congestion at the IP level? The issue at stake here is that congestion is a problem of queues filling up trying to save packets in front of a bottleneck, whether this bottleneck is at the IP level or not. It is well-known that layered abstractions provide a useful tool for the design of modular and inter-operable hardware and software, but are much less relevant concerning implementation and performance issues [14]. A flow control scheme purely designed at the IP layer would consider an abstracted lower, link layer. But congestion and drops may also happen at that lower layer, either in switches or in interfaces cards. Overflown queues dropping packets do not care about their OSI layer; they care about the amount of free buffers they have and about neighbour queues (upstream and downstream). Neighbour queues are generally *not* queues at the same layer, which is why congestion control is hard to implement layer-wise [15][16].

For instance, preventing sockets (level 4) from overflowing the transmission ring of the Ethernet driver (level 2) *cannot* be achieved according to some *layered* flow/congestion control scheme: a backpressure signal must generally cross a layer boundary. Symmetrically, the IP stack of a receiver overflown by Ethernet packets has no choice but to back-pressure its Ethernet device which is located at a different layer. *Every* queue must exert back-pressure upstream if one wants to avoid packet losses, and producer and consumer pairs are usually located at *different* levels. The only case where the upstream and downstream queues are located at the same level is when this level is the lowest. It is much more often Ethernet than IP. However, this does not prevent layer-wise flow control (like TCP receive window for instance) to provide *useful optimizations*: there is no point in injecting packets in the network if we know in advance that the receiver will drop or back-pressure them.

IV. NoQ proposal

A. Implementation

The basic idea behind back-pressure is simple (not all of its implications are). A congested queue signals its state to upstream queues and thus delay the delivery of additional packets, until it has regained enough buffers to avoid dropping incoming traffic. This flow control signal may then propagate backwards if needed, in order to prevent any loss of packets. We are currently trying to implement a fully back-pressured *Network of Queues* (NoQ) by slightly modifying Linux TCP/IP networking software. Let us review the modifications needed.

Transport protocols: to be adapted to NoQ, they will require very little modifications compared to current IP-based transport protocols implementations: this implies only the modification of the congestion related code, suppressing the deleterious slow start and congestion window algorithms on this NoQ interface, as we have successfully demonstrated with the TCP implementation of Linux 2.4 in section II-D[2].

Queueing inside nodes: Actually, back-pressure is already implemented *inside* network nodes; most of them usually do not drop packets internally, they only drop *incoming* packets when their corresponding input buffer is full. For some reason (small latency? trust?), network nodes are used to drop packets coming from outside, but not from inside. Figure 8 is an interesting example of this: since packets from different TCP flows are not dropped but blocked inside the host, TCP's fairness (RFC2581), based on drops, is technically *not* implemented inside the Linux 2.4.20 kernel. Of course, since packets are usually dropped just later in the "outside" network, this is usually a non-issue.

Linux receivers: Thanks to the new NAPI [17] ethernet driver design, making a Linux receiver able to back-pressure the network has been straight-forward.

IP *to* IP *flow control:* has to be implemented for cases where there is no lower layer, like on IP/DWDM links for instance.

B. Benefits

Throughput: Transmitting nodes are no longer limited by a conservative congestion window, they can instantaneously transmit as fast as they can. They will ultimately be throttled down to the capacity of the network bottleneck on the path they use, which will "back-pressure" them. The throughput of the protocol no longer depends on link peculiarities (*e.g.*, burstiness) or on constants like Maximum Transmission Units, and packet rate is aligned on the wire-rate of bottlenecks. Moreover, throughput no longer depends on latency, as long as buffering is provided.

Fairness: TCP flows try to fairly share bottlenecks [3]. In NoQ, fairness would not be among flows anymore but more likely implemented among "ports", thanks to round robin scheduling for instance (NoQ does not prevent more complex packet scheduling strategies). Fairness can converge instantly, since it is implemented right at the multiplexing point instead of end to end.

Efficient usage of buffers: In TCP/IP networks, only buffers of congested links are used; all other buffers are wasted, since links with a capacity greater than the bottlenecks keep their queue empty. Thanks to its backwards propagation of congestion, NoQ may effectively use *all* buffers on the route to damper transient congestion and avoid packet losses.

C. Issues

Of course, all this does not come free. The concept of back-pressure is not new, there are simple reasons why it is was avoided in the Internet, for instance.

Latency: Back-pressure makes it impossible to control latency (there is a throughput/latency tradeoff here). Let us underline again that NoQ is *not* compatible with time-sensitive applications like multimedia or interactive applications; it is clearly designed at solving specific issues like reliable high speed file transfers in some computing grids. A NoQ should admittedly be implemented along a classical TCP/IP network which would serve as a control plane for it.

HOL blocking: The main issue with hop by hop congestion control is definitely Head Of Line (HOL) blocking: by filling up queues, a congesting flow may hurt other flows sharing the same buffers, since the buffers cannot tell the difference between flows. This is especially disappointing when others flows have a different, non-congested ultimate destination[3]. There is no perfect solution to HOL blocking, but a broad range of techniques to mitigate it, that can be used cumulatively.

HOL blocking is mainly due to the logical multiplexing of flows [18]. Logical multiplexing is for instance when an IP queue cannot distinguish different TCP flows, or an Ethernet queue can not distinguish among IP destinations. This prevents any differentiation in resource management. The first, obvious way to mitigate HOL blocking is to limit logical multiplexing by limiting the number of concurrent flows: in the case of small, dedicated grid networks with only a few flows at once carried by a few $MP\lambda S$ paths, a NoQ will be applicable. On the other hand, HOL blocking makes NoQ irrelevant in huge networks. The threshold has to be investigated.

Another way to limit logical multiplexing and mitigate HOLB is to have more complex switches, with a more refined understanding of flows. For instance switches implementing Virtual Output Queueing [19], [20] or even circuit-oriented techniques. Again, this solutions scales to a limited, but still useful extent.

Some other receiver-based solutions allow limiting congestion as a whole:

- balanced network engineering helps avoid as often as possible cases like "1 Gb/s sender to 100 Mb/s receiver";
- message passing, as opposed to stream-based APIs, allow the receiver to know in advance the size and destination

[2]The code is available at `http://marc.herbert.free.fr/noq/`

[3]Considering a similar traffic problem transposed in a standard TCP/IP network, the congesting flow will also harm other flows; but the main difference is that congestion will not propagate backwards.

of expected data, thus avoiding slow and unpredictable memory management operations;
- additional flow controls that *bypass* some queues and direct queue-to-queue controls, for instance a tuned TCP's end to end receive window, provide useful optimizations: some network location may refrain itself from sending too much data, knowing in advance that a bottleneck in front of some far-away target would be congested.

V. Conclusion

Our experiments have first demonstrated the potential impact of the link layer behaviour on TCP performance, then how Ethernet flow control can break this dependence and bring wire-rate performance and fairness to applications. These experiments also taught us about the risk of too-abstracted models, and the fact that every buffer in the system must be taken into account when considering packet-loss issues. Finally, we have elaborated and studied feasibility of a fully back-pressured network (NoQ), that can be implemented thanks to slight modifications to existing networking hardware and software. We think that the main issue of a back-pressured network (Head Of Line blocking) can be moderated in the specific context of grid networks of limited size.

Acknowledgements

Marc Herbert and Pascale Primet are members of the INRIA team RESO, located in laboratoire LIP in Lyon. Marc Herbert is co-sponsored by SunLabs Europe located in Grenoble. The authors thank Benjamin Gaidioz and Gabriel Montenegro for invaluable discussions and Olivier Glück and Alex Cannara for attentive proof-reading of drafts.

Besides the ones already named above, these experiments could not have been produced without the following software: `tcptrace.org` and `xplot`.

References

[1] D. D. Clark, "The design philosophy of the DARPA internet protocols," *SIGCOMM Computer Communication Review*, vol. 18, pp. 106–114, Aug. 1988. [Online]. Available: http://doi.acm.org/10.1145/52324.52336

[2] K. K. Ramakrishnan, S. Floyd, and D. L. Black, "The addition of explicit congestion notification (ECN) to IP," Sept. 2001, standards Track. [Online]. Available: http://www.ietf.org/rfc/rfc3168.txt

[3] V. Jacobson and M. Karels, "Congestion avoidance and control," in *Proceedings of SIGCOMM'88*, Stanford, CA, Aug. 1988.

[4] T. J. Hacker and B. D. Athey, "The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network," in *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, Apr. 2002.

[5] S. Floyd, "RFC 3649: Highspeed TCP for large congestion windows," Dec. 2003, experimental. [Online]. Available: http://www.ietf.org/rfc/rfc3649.txt

[6] T. Kelly, "Scalable TCP: improving performance in highspeed wide area networks," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 2, pp. 83–91, Apr. 2003. [Online]. Available: http://doi.acm.org/10.1145/956981.956989

[7] L. Brakmo and L. Peterson, "TCP vegas: End to end congestion avoidance on a global internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[8] F. Paganini, Z. Wang, S. Low, and J. Doyle, "A new TCP/AQM for stable operation in fast networks," in *Proceedings of INFOCOM*. San Francisco: IEEE, Apr. 2003.

[9] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proc. of SIGCOMM*. ACM, Aug. 2002, pp. 89–102. [Online]. Available: http://doi.acm.org/10.1145/633025.633035

[10] IEEE, Ed., *802.3 Standard*. IEEE, 2002, ch. 2, annex 31B: MAC Control PAUSE operation. [Online]. Available: http://standards.ieee.org/getieee802/802.3.html

[11] R. Seifert, *Gigabit Ethernet*. Addison Wesley, Aug. 1999.

[12] M. Carson and D. Santay, "NIST Net – a linux-based network emulation tool," *Computer Communication Review*, to appear.

[13] R. S. Prasad, M. Jain, and C. Dovrolis, "On the effectiveness of delay-based congestion avoidance," in *Second International Workshop on Protocols for Fast Long-Distance Networks*, Feb. 2004. [Online]. Available: http://www-didc.lbl.gov/PFLDnet2004/

[14] D. D. Clark and D. Tennenhouse, "Architectural considerations for a new generation of protocols," in *Proceedings of the ACM symposium on Communications architectures & protocols*. ACM Press, Sept. 1990, pp. 200–208. [Online]. Available: http://doi.acm.org/10.1145/99508.99553

[15] T. Moors, "A critical review of "end-to-end arguments in system design"," in *Proc. International Conference on Communications (ICC)*, Apr. 2002. [Online]. Available: http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=21515&page=7

[16] D. Duchamp, "Alternatives to end-to-end congestion control," in *3rd New York Metro Area Networking Workshop*, Sept. 2003.

[17] J. Hadi Salim, R. Olsson, and A. Kuznetsov, "Beyond softnet," in *5th Annual Linux Showcase & Conference (ALS '01)*. USENIX, Nov. 2001. [Online]. Available: http://www.usenix.org/publications/

[18] D. C. Feldmeier, "A framework of architectural concepts for high-speed communication systems," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 4, pp. 480–488, May 1993. [Online]. Available: http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=5782

[19] W. J. Dally, "Virtual-channel flow control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, Mar. 1992. [Online]. Available: http://ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=127260

[20] Y. Tamir and G. L. Frazier, "Dynamically-allocated multi-queue buffers for VLSI communication switches," *IEEE Transactions on computers*, vol. 41, no. 6, pp. 725–737, June 1992. [Online]. Available: http://ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=144624