Policing Freedom
to Use the Internet Resource Pool

Arnaud Jacquet
BT

Bob Briscoe
BT and UCL

Toby Moncaster
BT

<firstname.lastname@BT.com>

ABSTRACT – ***Ideally, everyone should be free to use as much of the Internet resource pool as they can take. But, whenever too much load meets too little capacity, everyone's freedoms collide. We show that attempts to isolate users from each other have corrosive side-effects - discouraging mutually beneficial ways of sharing the resource pool and harming the Internet's evolvability. We describe an unusual form of traffic policing which only pushes back against those who use their freedom to limit the freedom of others. This offers a vision of how much better the Internet could be. But there are subtle aspects missing from the current Internet architecture that prevent this form of policing being deployed. This paper aims to shift the research agenda onto those issues, and away from earlier attempts to isolate users from each other.***

1. INTRODUCTION

Resource pooling allows separate resources to appear as one larger resource, giving resilience and efficiency gains. Recently it has been noted [1] that the history of communications has seen advances in resource pooling over more and more dimensions.

- Packet switching allows each link to act as a pool for packets from many separate sessions instead of needing one circuit per session.
- Packets can be time-shifted into a continuous time pool by utilising a buffer at a link rather than using time slots.
- Multipath routing pools together separate links into one network resource [9].
- Network coding pools multiple messages into fewer messages over a pool of links [8].
- With swarming downloads (e.g. BitTorrent) each receiver pools multiple peers sending the same data into one data source; and it pools the network paths from these peers.

By design, the public Internet gives everyone the freedom to use as large a share as they can take of any network equipment in the world, as often as they want. It is the classic pooled (or cloud) resource. This freedom has fostered an amazing array of inventive new uses for computers and communications.

Although pooling can make maximal use of available resources, congestion still results if too much pooled load meets too little pooled capacity. When any one customer's freedom to use the pool starts to limit the freedom of others we have no principled way to resolve the resulting conflicts. As we shall see, the cause turns out to be a subtle lack of architectural support.

In the eighties, over provisioning and voluntary restraint allowed the problem of resource accountability to be last in the list of requirements for the Internet architecture [4]. But, in today's primarily commercial deployment, lack of architectural support for resolving conflicts over pooled resources is fuelling an 'arms race'.

ISPs want to prevent a few customers hogging the whole resource pool when it is congested. Otherwise everyone assumes the ISP has grossly under-supplied capacity. So as each new way is found to pool resources, new piecemeal constraints are invented to 'unpool' usage into pieces that ISPs know how to control (we survey the most influential approaches in §3). This becomes a vicious cycle and it is becoming increasingly hard to invent a new use for the Internet that can also pick its way through the trail of throttles and blocks resulting from this arms race. Everyone's freedom to shift around what should be a general purpose resource pool is gradually being stifled.

Everyone's usage should be able to range freely over all dimensions of the resource pool and only be constrained when they restrict the freedom of others. Any usage should also be free *not* to yield to other usage, but the pressure to yield should increase the more congestion it experiences, and the longer this congestion persists. We are not saying ISPs ought to provide such an unconstrained service, but the architecture shouldn't prevent them doing so.

To this end, this paper uses a familiar conceptual device – a token bucket – but in an unfamiliar way. It controls the '*congestion* bit rate' of a customer, rather than their *actual* bit rate. Readers will be familiar with token buckets that discard packets that locally exceed the token rate and burst size. Instead the proposed policer solely counts the subset of packets that are congestion marked. Its token rate and burst size then places an overall limit on how much congestion packets traversing it can cause *everywhere else* – when all taken together regardless of flows.

In the body of the paper we give an example where one customer accesses multiple remote sites using multiple flows, with all congestion-marked packets being counted by the congestion policer. As congestion rises for a subset of the flows, the policer makes it advantageous to have a

All authors are partly funded by Trilogy, a research project (ICT-216372) supported by the European Community. The views expressed here are those of the author(s) only.

transport that shifts more traffic onto less congested paths [1], which is feasible if all the remote sites are serving the same data (e.g. using BitTorrent) or multiple paths are being used to the same service. If however some flows can only use those paths with rising congestion, the policer gives them no choice but to reduce their rate.

Thus, as well as limiting the total cost (congestion) that one customer can cause others, this simple bulk policer ensures each flow exhibits a dynamic response to congestion, but without policing each flow. Not all applications have to respond quickly to congestion, as long as the overall response is sufficient. If it isn't, the policer will eventually force even inelastic flows to terminate.

Note that we are not proposing this policer as an essential piece of the Internet architecture. It is a simple (albeit effective) example of what ISPs ought to be able to do, but currently cannot because the Internet architecture prevents them. The aim is to use this policer to focus the research agenda on the architectural changes needed to resolve conflicts within the Internet resource 'cloud'.

Next we present the target architecture we believe is required for congestion policing to be feasible. Then we place our work in the context of other influential ways to police network traffic. In §4 we introduce the details of the policer itself. In §5, we describe how policing congestion in bulk encourages individual flows to respond to congestion. We defer discussion to §6, before concluding.

2. TARGET ARCHITECTURE

- We avoid locating any mechanism at network resources themselves for resolving usage conflicts. Otherwise, each resource would have to check how much each customer was using every other resource in the pool;
- Instead, policing is located at the 'enforcement point' where a customer attaches (Fig.1);
- To police congestion experienced elsewhere, we need remote congestion to be visible in network layer headers;
- To do this we need every resource to randomly mark packets as it approaches congestion (explicit congestion notification or ECN [13]);

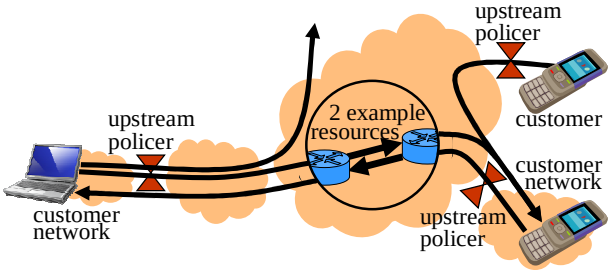

Fig. 1 – A set of congestion policers protecting all the entrances to a resource pool

- ECN reveals remote congestion to policers at the exits of the resource pool. Policers at the entrances can see remote congestion by making packet delivery conditional on the sender also marking packets with the congestion it expects on the rest of the path (re-ECN [3]).
- To be concrete, we assume policing at every entrance (Fig 1), then §6.4. compares it with policing on exit..
- The literature [3] explains how to use congestion revealed in packets for inter-domain traffic contracts;
- Only *the overall traffic* of each customer is policed;
- Making individual flows conform to a given congestion response is a non-goal; it would prevent evolution of perfectly reasonable new behaviours that shift the duty of responding to congestion around the pool of usage;
- There is no need to identify or to trust remote endpoint identifiers like source IP addresses and port numbers. Even if virtual customers share the same physical attachment, only locally assigned identifiers need to be trusted (link-local IDs like L2TP or VPNs).

This architecture polices packets not flows. It can only do this by adding information about remote congestion to network layer headers, which makes each packet sufficiently self-contained to be accountable. This is feasible because congestion is a property of every bit in each packet, unlike bit-rate which is meaningless below flow granularity. Once packets are accountable, in turn those who transfer them into (or out of) the resource pool can be made accountable–locally at their attachment point.

This architecture currently only improves resource pooling for ECN-capable packets. This places a new question on the research agenda: To support resource accountability should the network include some aspect of loss detection, rather than leaving it solely to endpoint transports?

3. RELATED WORK

Table 1 displays some of the most influential approaches for policing Internet resource usage, arranged along a spectrum that characterises how flexible each approach is to resource pooling. Each is further characterised by what 'Metric' it uses to judge excessive usage and what 'Constraint' it places on usage when the metric rises.

In terms of flexibility to pool resources, the critical feature of each approach is the *granularity* at which it applies metering and policing ('/customer', '/source', '/link', etc). The term 'per customer' doesn't mean 'per user' but 'per locally attached contractual entity'. The customer relationship need not be commercial. The term customer includes customers with many users. 'Per link' means that usage conflicts are resolved at each bottleneck link (not necessarily all links, just those likely to congest). For brevity, we will only discuss some rows of the table.

Voluntary restraint is exercised in the current Internet by application developers who choose to use a congestion responsive transport like TCP. In an unpublished 'survey of 14 surveys' (2003-6) we found that TCP comprises 73%-94% of traffic. But some customers can run more TCP sessions than others and for much more of the time, so it is a fallacy that prevalent use of TCP implies anything about

fair, equitable sharing of capacity. It implies only that a voluntary *dynamic* response to congestion is still prevalent.

Name	Metric	Constraint	Flexibility
voluntary restraint	rate /customer	peak rate /customer	↑ free
Congestion pricing	congestion /customer	pricing /customer	
Congestion policing		**congestion rate-burst /customer**	
vol. pricing	(peak period) volume /customer	pricing /customer	
volume cap		rate cap /customer	
deep packet inspection		rate cap /customer /app type	
(W)FQ	Packet presence /source /link	(weighted) equal rate share /source /link	constrained ↓
(W)FQ	*-ditto- but* /flow /link	*-ditto- but* /flow /link	both & neither
bottleneck flow policer	rate share /flow /link	rate cap /flow /link	

Table 1 – Spectrum of policing approaches

Congestion pricing involves deployment of ECN then charging for the volume of packets marked 'congestion experienced'. It is structurally very similar to the congestion policing scheme of this paper. Gibbens & Kelly wanted to allow applications to evolve without specific constraints on how they should respond to congestion, but within an overall economic incentive to cause no more congestion than you would be prepared to pay for [7].

Congestion policing is the focus of this paper. We show it can provide the same incentives as dynamic congestion pricing with the same clean engineering simplicity. But we want ISPs to be free in their choice of pricing model, including flat fees. So we follow the advice of Odlyzko, who gathered evidence across many market sectors to show that, on short timescales, customers prefer rationing of supply to dynamic variation in price [11]. Congestion pricing gives people *too much* freedom – they worry they will spend more than they have budgeted for.

Volume is used as the metric in the next three rows of the table. Unlike congestion, using volume doesn't produce particularly correct incentives, but it is currently a pragmatic alternative, given ECN is not widely deployed and the Internet architecture wasn't designed for remotely detecting losses in the network layer.

Counting only peak volume would better approximate congestion. However, this still counts traffic on uncongested paths as much as on congested paths. Also, the majority of traffic in a network is in the large transfers, but accounting for volume gives them no incentive to back away during peaks in congestion, whereas accounting directly for congestion does. If a large transfer gave short flows the space to go faster they would finish much earlier, freeing up capacity sooner for the long-running traffic. So accounting for congestion makes the network appear very fast for interactive traffic without affecting the completion time of larger transfers. Whereas accounting for volume gives no incentive for traffic to re-arrange itself along the time dimension of the resource pool.

Fair queuing (FQ) [10], which may be weighted (**WFQ**), divides up a link's instantaneous rate into equal (or weighted) proportions, but only among those sources that are currently sending. So if there are 200 possible sources but only 5 are active, they will get 1/5 of the capacity. If some of the active sources send a lot more than others, (W)FQ grows their own queue rather than encroaching on the shares of others. But, if more sources become active, (W)FQ reduces everyone's share – in our example it guarantees each source 1/200 in the worst case.

Sources that are infrequently active share a link much more efficiently than high activity factor sources, but (W)FQ gives no credit for a low activity factor. Prior inactivity entitles you to no more than if you had been permanently active. By not recognising the time dimension of the resource pool, (W)FQ doesn't encourage activity to shift in time to avoid other activity, even though a lot of applications can do this, either shifting by seconds (buffering) or by hours (e.g. into the night).

Of course, the original goal of FQ was to isolate users from each other. We are not implying this is never a worthwhile goal. Our aim is merely to highlight that isolation hides opportunities for valuable co-operation.

Two proposed approaches at **per flow granularity** are shown at the bottom of the table. Demers *et al* proposed that fair queuing would be more appropriate per flow than per source, as large sources (e.g. servers) deserved a greater share [5]. However, the authors recognised that any host could increase its link share without bound by opening connections with multiple other endpoints.

Despite this well-known flaw, research proposals to police on a per-flow basis have continued [6, 12]. These per-flow approaches constrain each flow too much and all flows too little. They proscribe reasonable transport designs where flows use the flexibility of the resource pool while doing nothing at run-time to prevent the set of a customer's flows overloading the resource pool [2].

4. POLICING CONGESTION

A system will encourage optimal behaviour if all the costs each individual causes others to suffer are reflected back on that individual. The challenge we set ourselves is to do this without unpredictable bills.

We set the constraint that the customer pays a flat monthly fee to its ISP. This funds a constant rate, *w*, of congestion tokens filling a bucket (Fig 2) – ISPs may give customers the choice between different values of *w*. Unlike with a classic token bucket, only congestion marked packets consume tokens. So tokens are not consumed based on the

amount of traffic sent, but on the amount of congestion the traffic causes, which ensures the customer suffers the cost of its behaviour on others. If the policed customer generates flows $i=1..N$, each of throughput $x_i(t)$ over a path experiencing congestion $p_i(t)$, the bucket empties at a rate $\sum p_i(t) \cdot x_i(t)$, which we will call the congestion bit rate (a classical token bucket would consume tokens at rate $\sum x_i(t)$). The depth β of the bucket allows the customer to cause bursts of congestion, allowing for fluctuations in network conditions, and in the customer's own needs.

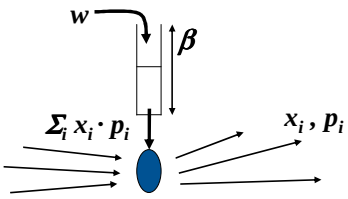

Fig. 2. A congestion policer can be implemented simply as a modified token bucket

So long as the customer stays below its congestion allowance, the policer merely monitors the congestion bit rate passively. But whenever the congestion bit rate empties the bucket, the policer penalises the aggregate it monitors. The penalty can take different forms: packets may be delayed or dropped. Dropping traffic keeps the congestion bit rate within the allowance and gives customers the strongest incentive to control their congestion bit rate themselves. The policer might transition smoothly from passive monitoring to actively penalising traffic. However, initial analysis suggests a smooth transition serves no purpose, though further research is required to confirm this.

5. IMPACT OF THE POLICER

5.1 Impact on the flows of the policed customer

One of the key features of the congestion policer is that it doesn't enforce a specific response to congestion per flow. Each flow is free to use any congestion response so long as all flows together don't cause the overall congestion bit rate of the customer to exceed the allowance given.

To explain, let's first imagine a policer configured to allow only an unrealistically low congestion bit-rate. Fig. 3 illustrates the effect of such a policer on a single long-running flow: either a TCP flow (congestion response: y_{TCP}) or a constant bit-rate flow (congestion response: y_{CBR}). If congestion increases enough, and the flow runs at a high enough rate for long enough, the sustained high congestion bit rate will empty the bucket. The policer will then override the congestion response of the flow so that the congestion bit rate equals the allowance ($p.y_{policed}(p)=w$) which gives the shape of the congestion response imposed by the policer: $y_{policed}(p)=w/p$.

The TCP flow has its own congestion response, $y_{TCP}(p)$ (for brevity we assume constant segment size and RTT). If the customer's allowance is low enough, it crosses the policer's congestion response at p_{TCP}. As long as $p<p_{TCP}$, the policer has no effect and the throughput is given by the TCP congestion response. If $p>p_{TCP}$ for long enough the policer starts to limit throughput to $y_{policed}$ instead.

The effect is even more significant for the unresponsive flow. Its throughput remains constant while $p<p_{CBR}$. But if $p>p_{CBR}$ for long enough, the policer makes the flow follow the congestion response defined by $y_{policed}$ too.

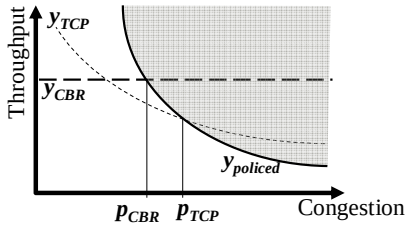

Fig. 3. If set harshly, the congestion policer would impose default congestion response $y_{policed}$ on single flows

More realistically, the allowance would be much higher to allow multiple flows of customer traffic. In the special case where all flows exhibited the same congestion response and shared the same path, the scenario would be equivalent but linearly scaled: a policer with 100 times higher allowance would start limiting 100 sustained TCP flows over a path with congestion p_{TCP}.

More generally, given the policer treats all flows in bulk, the congestion bit rate of each flow has the potential to affect the throughput of all others. In order to see how this encourages more elastic flows to compensate for less elastic flows, we must first quantify the cross-flow interaction the policer introduces.

The unpoliced congestion bit rate generated by a customer is $v = \Sigma p_i \cdot y_i(p_i)$, where y_i is the congestion response of flow i's data rate. The policer starts to intervene if $v>w$ for long enough. The penalty imposed to ensure v doesn't exceed w is the same on all packets. Thus the policer increases the congestion level for all flows by the same amount π.

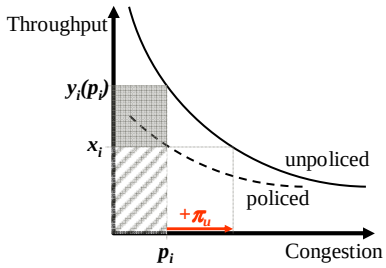

Fig. 4. Effect of bulk policing on a responsive flow

Fig. 4 shows how the throughput of a responsive flow is reduced to $x_i = y_i(p_i + \pi)$, based on its congestion response. TCP flows in steady state have a congestion response proportional to $1/\sqrt{p}$. So, for small values of π, linear approximation gives the flow's throughput as:

$$x_i = y_i(p_i + \pi) \sim y_i(p_i) + \pi \cdot y_i'(p_i + \pi) = y_i(p_i) \cdot (1 - \pi/(2p_i))$$

The p_i in the denominator shows that the congestion policer has the greatest effect on flows on the least congested paths. Customers aiming to maximise their total throughput thus have the incentive to take charge of congestion control across all their flows (esp. those on the most congested paths), to make the most of their congestion allowance.

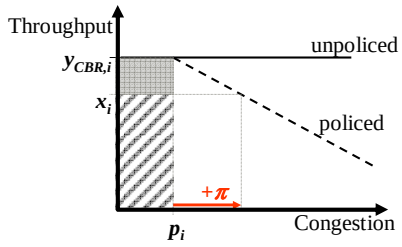

Fig. 5. Effect of bulk policing on an unresponsive flow

Using the same analysis, Fig. 5 illustrates how the policer forces unresponsive flows to respond to congestion, reducing throughput from $y_{CBR,i}$ to $x_i = y_{CBR,i} \cdot(1- \pi)$. The congestion bit rate of the flow is reduced from the larger grey area to the smaller hatched area.

If a customer has a mix of elastic and CBR traffic the policer allows the customer to let its CBR flows remain unresponsive through heavy congestion. At the same time, it causes packet drops that force elastic flows to compensate. These extra drops should encourage evolution of host optimisation software that ensures elastic flows compensate in advance by reacting more strongly to congestion, as the customer's overall traffic approaches the policed allowance. Effectively, elastic transports don't have to be modified for the policer to make them self-sacrifice to CBR flows, but if they do, the customer is better off.

5.2 Impact on cross traffic

Each customer's traffic clearly benefits if the congestion bit-rate of competing cross-traffic is limited. Indeed, every other customer benefits when sources of heavy congestion are pressured to shift to less congested bottlenecks and if they can be prevented from contributing excessive congestion, across one or several parallel bottlenecks.

6. DISCUSSION

6.1 Why per flow responsiveness is not enough

Imagine customer *A* uses 98 shares of a bottleneck and customer *B* uses 2. If the bottleneck loses half its capacity, they can both halve their usage to prevent congestion. However, *A* still has 49 times more than *B*. *A* might have opened 98 TCPs while *B* opened only 2. Or *A* might keep 2 TCPs active 49 times more often.

Therefore prevalent per flow congestion responsiveness certainly prevents congestion collapse. But it is a fallacy that it *also* controls fair sharing of resources, particularly given it ignores sharing over time.

However, our congestion policer shows the reverse approach is fruitful. Ensuring fair shares of everyone's overall congestion contribution *does* also ensure per flow congestion responsiveness – both voluntarily and ultimately by enforced intervention (§5).

6.2 Endpoint evolution driven by congestion policing

Policing a customers' overall congestion is designed to make their own congestion control evolve beneficially. If the policing is triggered by the customer's overall behaviour, it doesn't discriminate between flows. So any valuable flows over uncongested paths will be unnecessarily throttled. Endpoint developers will be driven by demanding users to create software to maximise the value they get out of the network under such a constraint. It is likely strategies will be found that minimise user intervention, so they can spread to the mass market.

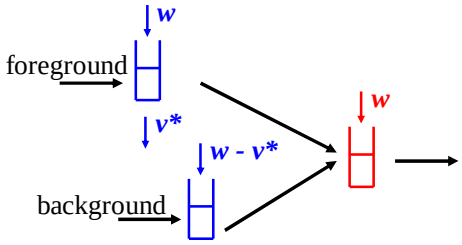

Fig. 6. A dual 'shadow' policer assists endpoints that can distinguish foreground and background traffic

For instance, software that can discriminate between foreground and background flows could benefit from using the dual "shadow" policer shown in Fig. 6. Foreground traffic is unconstrained by this policer but its congestion bit rate v^* is monitored (as long as it remains within the overall allowance). Meanwhile, background traffic is more severely constrained by another congestion policer, against the surplus allowance w-v^* unused by foreground traffic.

Currently some operators are using deep packet inspection to prioritise packets by their payload. The example above shows that we actually only need a bulk constraint on congestion in the network to encourage application discrimination to evolve where it is more appropriate – on endpoints.

6.3 Choice of Bucket Depth

The bucket depth gives the extent to which sporadic customers can 'roll over' the congestion allowance gained during idle periods. It also defines the maximum congestion burst a customer may cause over the network.

Congestion would be more predictably smooth if everyone were limited to smaller bucket depths, but this would reduce flexibility to make urgent demands on the network. Further research is planned to formalise this trade-off.

6.4 Policing upstream or downstream?

The same token bucket-based contract could be used to either police a customer's upstream (outgoing) as in Fig. 1,

or downstream (incoming) traffic - or both. However, the choice has profound architectural implications.

A strong argument can be made for policing upstream traffic, so that it can be limited before it does any damage. Also, at the network layer, the sender is ultimately responsible for sending excess traffic into the network. Certainly the receiver may have originally requested the connection (and blame might be traced back further), but that is all strictly above the scope of the packet forwarding layer that a policer protects. However, the policer we have described relies on counting congestion markings in the packets it handles, and at the ingress, they haven't yet traversed any congestion. The re-ECN protocol [3], which forces the sender to mark expected congestion into packets, is a proposed solution to this problem.

On the other hand, ECN already makes it straightforward for the policer to count congestion information arriving in downstream traffic. Discarding downstream traffic doesn't necessarily stop the sender continuing to cause congestion, but it does stop the receiver getting the data. In most cases, except deliberate malice, the receiver's feedback loop would have the desired effect of slowing the sender to the policed rate. However, any sender could run down the receiver's token bucket with unsolicited traffic, so policing downstream traffic would open a new DoS vulnerability.

We need to make it clear that, if network A deploys the policer we have proposed at its customer's attachment point, it doesn't only police congestion experienced in network A. It is intended to count congestion experienced all along the path in other networks. Nonetheless, at the start of incremental deployment, a network gains from having policers deployed for both traffic directions.

7. CONCLUSION

We lack principled ways for ISPs to prevent customers over-using a 'cloud' service like the Internet. ISPs often use ad hoc techniques to isolate users from the adverse effects of others. But users' software then lacks adequate signals and incentives to shift traffic around the resource pool - to less congested links or times. As fast as advances in resource pooling are being invented, these over-restrictive ad hoc controls are undermining them.

We propose a general purpose traffic conditioning contract that is more appropriate for a cloud service than these rate and volume-based alternatives. It can be implemented by a simple token bucket at the customer's attachment point, which limits the customer's total contribution to congestion anywhere in the resource pool.

Without imposing any particular behaviour on individual flows, this policer encourages flows to move to less congested paths and to respond to congestion on their own path. If they don't, it protects the freedoms of others using the resource pool by forcing a response across all flows. But the imposed response will always be worse than the customer's own software taking charge of each flow's separate response.

The proposed alternative of imposing a response on each flow certainly prevents congestion collapse but it wouldn't control resource sharing as claimed and it restricts evolution of new transport behaviours.

We believe the architectural agenda should shift from promoting per-flow controls to improving information sufficiency in packet headers. We ask what information would make packets sufficiently self-contained to be held accountable for the congestion they contribute to. We have shown that ECN may be sufficient, but open questions remain concerning whether resource accountability also requires prevailing loss to be revealed in packet headers, and whether rest-of-path congestion is needed.

8. REFERENCES

[1] Anonymous. "A resource-pooling architecture for the Internet" Under submission, Jul 2008

[2] B.Briscoe. Flow Rate Fairness: Dismantling a Religion, ACM CCR 37(2) 63--74 (Apr 2007).

[3] B. Briscoe, A. Jacquet, T. Moncaster and A Smith. Re-ECN: Adding Accountability for Causing Congestion to TCP/IP. IETF draft-briscoe-tsvwg-re-ecn-tcp-06. July 2008 (work in progress)

[4] D. Clark, "The design philosophy of the DARPA internet protocols," Proc. ACM SIGCOMM'88, Aug'88.

[5] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in ACM Symposium proc. on Communications architectures & protocols. 1989

[6] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the Internet. IEEE/ACM ToN, Aug. 99

[7] R.J. Gibbens and F.P. Kelly. Resource pricing and the evolution of congestion control. Automatica 35 (1999)

[8] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard and J. Crowcroft, "XORs in the Air: Practical Wireless Network Coding" Proc. ACM SIGCOMM'06, (September, 2006)

[9] C.N. Laws. Resource Pooling in Queueing Networks with Dynamic Routing. Advances in Applied Probability. September 1992.

[10] J. Nagle "On packet switches with infinite storage," IETF RFC970. December 1985.

[11] A. M. Odlyzko, Paris Metro Pricing for the Internet, Proc. ACM Conference on Electronic Commerce (EC'99), 1999

[12] R. Pan, L. Breslau, B. Prabhaker, and S. Shenker. "Approximate fairness through differential dropping,". ACM CCR 33(2), April 2003.

[13] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, Sept. 2001.