Coordinated Bandwidth Control in Data Centers

Krishna Kant

Intel Corporation

Abstract

TCP/IP over Ethernet is well established fabric in data centers and is expected to play the role of a "unified fabric" due to emerging technological and application trends. QoS and congestion performance of such a fabric is critical to its success. It is well known that proper IP QoS setup is very complex and is either ignored completely or set rather simplistically. Unfortunately, without an elaborate end to end QoS setup, TCP connections simply divide up the available excess bandwidth equally among themselves under congestion. This is not necessarily desirable in a data center. In this paper we propose autonomic mechanisms that determine BW requirements of various flows and provide bandwidths to them in proportion to their requirements. The estimations are done dynamically and thus can easily track changing application requirements. The paper shows that the scheme not only yields close to desired bandwidth allocation, but also significantly reduces packet losses.

Keywords: Cluster computing, TCP, QoS, congestion control, collective control, diffserv

1 Introduction

Modern data centers have grown to be large networked environments with thousands of servers and tens to hundreds of different applications. Most current data centers use SMPs (symmetric multiprocessors) for large applications like databases, centralized SAN (system area network) storage (usually Fiber Channel based), and, if needed, specialized fabrics for low-latency inter-process communication (IPC). However, there is a strong emerging trend from this traditional picture towards a cluster computing environment built around a single *unified fabric* that carries all traffic types. (See [2, 1] for more complete discussion on motivations for this trend.) There are, of course, several fabrics that are well suited to data center

communications, e.g., Infiniband [IBA] [3], QSnet (www.quadrics.com), Myrinet (www.myricom.com), etc. However, the pervasiveness of Ethernet (and in fact that of TCP/UDP over IP over Ethernet) ensures that other fabrics will remain niche. In fact, Ethernet[1] becomes a natural *unified fabric* when we consider developments such as 10 Gb/sec Ethernet data rates, HW offload and user mode implementation of TCP/IP [4], iSCSI (SCSI protocol over TCP/IP), etc.

Ethernet as a unified fabric demands appropriate quality-of-service (QoS) features to cater to at least the most common traffic types within the data center. The primary role of QoS here is to ensure that the major *applications* can maintain a good performance under stress situations and are able to provide service differentiation as mandated by the SLAs (service-level agreements). The emphasis on applications is paramount and a distinguishing feature. In particular, each application may involve many types of traffic "flows" with different absolute or relative QoS requirements for them. Typically, within a data center, each traffic "flow" is carried over a separate TCP connection (e.g., client-server connections, IPC connections, storage connections, etc.). Since each TCP connection inherently behaves independent of others, it is very difficult to make them behave cooperatively under congestion situations.

Apart from TCP behavior, the other main hurdle to data center QoS is the need to do complex setup at not only the endpoints but also the intermediate routers and switches. The setups require detailed knowledge of the application behavior and requirements, which is generally not available. In fact, the data center administrator knows little about the applications being run and often simply can't afford the time and effort required to do the setup. Furthermore, even if an initial setup could be done well, most applications show a wide range of dynamic behavior and evolve; therefore, an elaborate initial setup could well be ineffective or even hurt performance. Instead, it is so much easier to simply overprovision resources and hope that problems arise very infrequently!

IP QoS is a good example of the difficulties mentioned above. IP QoS is very well developed and provides a rich arsenal of techniques including:

1. End to end bandwidth reservation (intserv) vs. per hop QoS (diffserv)

[1]We regard everything from transport layer downwards as the *fabric* and thus Ethernet as a fabric is really used in the sense of TCP/UDP over IP over Ethernet.

2. Queuing schemes, e.g., priority queuing, weighted fair queuing, etc.)

3. Packet drop schemes, e.g., tail drop, random early drop (RED), weighted RED, feedback RED, etc.

4. Traffic policing and shaping (e.g., leaky bucket based schemes).

Given a specific set of applications, a good static characterization of their behavior and their QoS needs, it is possible for the most part to use these schemes to set up just the right kind of treatment for each flow. In particular, at each endpoint and router, we could decide on the bandwidth, queue size, queue scheduling policy, drop threshold, drop percentage, etc. for each flow. However, as discussed above, this is simply not practical and rarely even attempted — a huge amount of research on the subject notwithstanding. Instead, much of the QoS parameters in reality are set in the following 3 ways: (a) left at their default or vendor-set values, e.g., RED vs. tail drop at a router, (b) set to "common sense" values, e.g., giving management control messages a high priority or treating streaming traffic better than data traffic, and (c) simple priority or threshold choices based on economic considerations (e.g., premium vs. normal service, important vs. normal customer, etc.).

We believe that to be successful, any manual QoS setup must remain "common-sense", and more sophisticated treatment should be autonomic in nature. In fact, the administrator should not have to specify any more about the application flows than their general type (e.g., real-time vs. non real-time, normal vs. premium, etc.). The autonomic mechanism should automatically learn the application characteristics, bandwidth requirements, etc. and then use some sort of a signaling mechanism to setup and evolve the QoS settings. Undoubtedly, this is a very tall order and we pretend not to know how feasible and successful it will ultimately be. We do, however, propose some simple schemes in this paper towards this goal and exhibit their performance.

The outline of the paper is as follows. In section 2 we discuss how data center environment differs from general Internet since much of the research on the subject to date is Internet focused. In section 3 we describe essential elements of collective bandwidth control mechanism and provide some insight into various decisions made in choosing the algorithms. Section 4 then discusses the experimental setup for evaluating the scheme and the results. Finally, section 5 concludes the discussion.

Before continuing, we briefly review the related work. TCP and TCP-like congestion control is an

extremely well researched area and we will not try to survey it. In particular, numerous loss and delay based schemes have been investigated of which the latter are particularly relevant for data centers [7, 6, 9]. Mathematical modeling of TCP behavior including fairness, stability, Nash equilibria, is another heavily studied area [11, 8, 10]. Enforcing weighted proportional fairness [12] is close to the ideas explored in this paper, except that our focus is on applications within a data center rather than individual flows with statically specified requirements.

2 Data Center Environment

Although much of the comments above about QoS setup problems apply to Internet in general, we specifically concentrate on data centers in this paper for two main reasons: (a) data centers have specific characteristics and needs that may not apply to Internet in general, and (b) the restricted data center environment allows us to exploit information and techniques that may be very difficult to implement in general. In this section we discuss these issues briefly.

A data center typically interfaces with the outside world through some special servers such as those dishing out built up web pages, email gateways, VoIP gateways, etc. Our focus in this paper is on the *inside* of the data center which accounts for increasing amount of computation before the response is sent back to the client. This domain is increasingly moving from SMPs to cluster computing and its attendant IPC (inter-process communication) which is often quite latency sensitive. The high cost of Fibre channel SANs is pushing storage towards a distributed IP-based model. For large transfers from fast disks, storage traffic can have pseudo real-time characteristics. Finally, with emerging real-time services such as VoIP and IPTV offered along with traditional data oriented services (e.g., web pages) mean even a streaming media component within a data center. Furthermore, the emerging notion of virtual clusters and utility data centers implies further mixing of various applications and traffic types on the same fabric.

The environment within a data center is generally quite different than in the Internet in general. Some of the key characteristics include: (a) much higher data rates, (b) much smaller and less variable round-trip times (RTTs), (c) low to very low end-to-end latency requirement, (d) less chances of severe congestion (compared with Internet in general), and (e) persistent connections and flows. Reference [2] discusses

consequences of these requirements in some detail. Here we address only the issues that are directly relevant to congestion control and QoS.

The high data rates generally imply an emphasis on simplicity (e.g., low CPU overhead) in protocol processing and care in congestion control so as to avoid big clumps of packet losses. The small RTT is helpful in congestion control in that congestion feedback and control can be effected in less than $\tilde{1}00\ \mu s$ (or lower with HW offloads, specialized switches, etc.). Low latency coupled with low processing overhead requirements imply that packet losses are particularly undesirable in a data center since they require traversing "slow path" in the protocol and may delay packets (lost packet and other already transmitted packets ahead of lost packet) substantially. Thus, one goal of congestion control in a data center is to minimize packet losses. We shall see later that our scheme does significantly better than the default TCP scheme.

Attribute (d) is a result of overprovisioning and congestion control at the data center edge. It means that *network congestion within a data center is more of an exception rather than a norm.* Attribute (e) generally results from the fact that flows/connections within the data center are driven by the application needs (e.g., IPC capability, storage access, etc.) rather than by individual client requests. We exploit these two attributes in our autonomic schemes for estimating bandwidth requirements of the application. In particular, we assume that the gap between successive congestion episodes is sufficiently long, the congestion duration is sufficiently small, and the flows sufficiently long lived so as to make autonomic estimation and control sensible in most cases.

From a QoS perspective, the major focus within a data center isn't an isolated traffic flow, but rather the entire application. This is a critical difference since an application can be considered as a set of *inter-related* flows with different characteristics and all these flows must get their requisite QoS in order for the application to make progress. For example, a clustered database application involves both IPC and storage traffic, the proportion of which depends on a variety of factors. Assuming a distributed iSCSI based storage, both traffic types will go over separate TCP connections between servers. If the IPC traffic is 5 times that of the storage traffic, equal division of available bandwidth between the two types simply will not work well. What is required that the same 5:1 ratio be maintained even under congestion situations. In other words, the celebrated "fairness" property of TCP is not desirable within

a datacenter.

Similar arguments can also be made between applications. For example consider two applications, say, database access and shopping cart. Presumably the two applications are connected in terms of their relative needs for processing e-business, so a drastic bandwidth cut for only one of them is not the right approach. Even if they are not connected, "fairness" and graceful degradation demands that all applications degrade by a similar factor. If SLAs or other factors make a different equilibria more desirable (e.g., "Essential" applications are squeezed only half as much as "normal" applications), we still need mechanisms that go beyond the default TCP behavior.

These considerations call for the notion of *collective bandwidth control* or more generally *collective resource control* so that all flows passing through a congestion point are controlled as a cooperating set, rather than as unrelated competing set. It is clear that such a collective bandwidth control needs to operate at the following two levels:

1. *Inter-application*, or comparative treatment of applications (e.g. BW subdivision) during periods of congestion, and

2. *Intra-application*, or comparative treatment of "flows" belonging to an application.

A *flow* is used here in the sense of type of traffic - for example, a DBMS application may be characterized as having 4 traffic flows: query/response (or networking), IPC control traffic, IPC data traffic and storage traffic. For the purposes of this paper, we further restrict a "flow" as a packet stream riding over a reliable connection such as TCP. It is possible to use the techniques described here at the application level for shaping datagram traffic (e.g., UDP) as well, but such a discussion is beyond the scope of this paper.

3 Collective Bandwidth Control

There are 3 essential aspects of collective bandwidth control: (a) determining all relevant flows (or connections in this paper) that need to be controlled together, (b) determining bandwidth requirements of such connections, and (c) starting and ending control. For simplicity, we henceforth assume that all

flows are permanent. The schemes can be easily extended to non-permanent flows provided that the flows are sufficiently long to make per flow monitoring sensible.

3.1 Determining Collective Control Set

The control set determination problem must be addressed independently by each server in the cluster since a coordinated determination, while theoretically feasible, would usually be too complex. The determination really involves estimating which connections from this server are affected by the existing congestion so that they could be controlled cooperatively. This is difficult to do purely from the default TCP behavior since there is no guarantee that all affected connections will receive ECN (explicit congestion notification) or experience packet loss. In fact, with a RED-like marking/dropping scheme, fatter connections are more likely to experience congestion indication.

The collective control set (CCS) can be identified in many ways, each with its pros and cons. In the following, we list a few methods and their properties. It is assumed here that all connections between the same pair of end-points generally follow the same path — a reasonable assumption within a data center.

1. **Performance Based**: In this scheme, the CCS changes dynamically based on the observed performance (retransmissions and high delay).

   This scheme is very general and does not require any infrastructure support, however, it suffers from two problems: (a) it lumps multiple unrelated congestion points into one, (b) determining the entire CCS may take some time, which gets in the way of effecting a quick control.

2. **Destination based**: If a TCP connection to destination X is found to be congested (due to high delay, ECN marking, or packet drop), the CCS consists of all connections to X from the server in question.

   The main virtue of this scheme is extreme simplicity – fixed and trivially determined CCS. The CCS here is correct if the congestion is at the destination or in the link connecting the destination; however, as the congestion moves away from the destination, the accuracy fades quickly. However, the CCS will never include unnecessary connections.

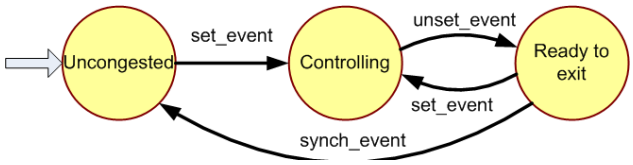

Fig 1: An illustration of bandwidth estimation process

3. **Destination router based**: This is a slight modification of the above scheme where router on which the destination homes in is assumed to be congested.

   This scheme requires knowledge of the topology, which is possible to obtain within a data center. Assuming shared buffering at the router (as opposed to per port buffering), this scheme can tackle congestion close to destination more effectively than the destination based scheme.

4. **ECN based**: In this case, we depend on changes to the ECN mechanism for revealing the congested router (or a congested outgoing link from a router which eventually results in backpressure at the router). Two possible approaches to ensure that every connection will learn about the router congestion are: (a) 100% ECN marking, or (b) intelligent ECN marking (based on source-destination IP addresses).

   This scheme is accurate but requires changes to normal router functioning. The scheme can be easily combined with destination based scheme to handle endpoint congestion.

Although these CCS schemes are under study, this paper is *not* focussed on this subject. Instead, the focus here is to show the usefulness of collective controls assuming that the CCS can be established accurately.

3.2 Operation of Control Cycle

As stated earlier, our scheme operates by autonomically estimating bandwidth requirements and other parameters of the connections and using them for collective resource control. At a high level, the control cycle involves the following three components:

(a) Detection of when the first member of a CCS starts encountering congestion so that the scheme can move from per connection control phase to collective control phase.

(b) Reliable estimation of the bandwidth and other relevant parameters (e.g., delay).

(c) Reliable estimation of when the CCS is free of congestion, so that a switchover back to individual control can be effected.

Fig. 1 shows these components in form of a generic state diagram. Initially, a connection is in the uncongested state and is running the estimation algorithms. The "set_event" refers to situations that result in the connection moving to a state where collective bandwidth control (CBC) goes into effect.[2] The "unset_event" refers to the situation where this connection does not see any congestion effects and thus declares itself ready to exit from CBC regime. However, it will still be subjected to CBC until all connections in CCS are ready to exit. The "synch_event" indicates this condition and makes the process start over.

3.2.1 Setting Controls

It is well known that TCP regulates flow of packets in response to two conditions: (a) explicit congestion indication via ECN or packet loss, and (b) increase in round-trip delay. Different versions of TCP may weight these events differently; for example, TCP vegas and the newer fast TCP [7, 6, 9] base their congestion control algorithm directly on the observed delays. Thus, "set_event" should also comprehend delay, ECN and loss. The last two are easy to handle: the CBC takes effect whenever TCP slashes window as a result of ECN processing, retransmission timer expiring, and fast retransmit. For the delay, we keep track of two quantities over the last N_{rtt} round-trip periods (where N_{rtt} is a parameter).

1. current_rtt. Average RTT over the last N_{rtt} measurements done by TCP. The averaging over multiple measurements is done to smooth out short-term variations.

2. base_rtt. This is an exponentially smoothed version of current_rtt with smoothing factor of α_r. This estimate is frozen as soon as the connection enters the "Controlling" state.

Given these estimates, the increase in delay is detected when ratio of current_rtt and base_rtt exceeds some threshold η. The threshold η must be chosen large enough so that CBC goes into effect only for

[2]Actually, the CBC will not do anything interesting until CCS size exceeds 1.

real congestion situations rather than due to variations in queuing and scheduling delays. For the results shown here, we conservatively chose $\eta = 1.4$.

The choice of the parameter N_{rtt} needs to balance two conflicting needs: (a) N_{rtt} should be rather small in order to detect RTT increase before the losses take place, and (b) N_{rtt} should be large enough to smooth out bumps due to packet losses. We find that a fairly small value of $N_{rtt} = 4$ is adequate and is used in the results. The parameter α_r needs to be rather small to allow a substantial difference between current_rtt and base_rtt as delays start to mount. At the same time, α_r should be large enough to produce a good estimate of RTT and bandwidth (see below) in a reasonable amount of time. In results, $\alpha_r = 0.05$.

3.2.2 Bandwidth Estimation

In many TCP implementations, the window size "cwnd" is often not a good indicator of used bandwidth; the bandwidth estimation was done directly. In particular, the following three quantities were estimated (this is similar in spirit to delay estimation discussed above):

1. current_bw. Estimated as the byte acknowledgement rate over the last N_{rtt} RTTs (using TCP's highest unacked sequence number).

2. reqd_bw. This is an exponentially smoothed version of current_bw with smoothing factor of α_r and indicates the required bandwidth. This estimate is frozen as soon as the connection enters the "Controlling" state.

3. avail_bw. This is also an exponentially smoothed version of current_bw but with smoothing factor of α_a. This estimate is updated continuously.

The parameter α_a must necessarily be significantly larger than α_r since the purpose of avail_bw to faithfully track the impact of ongoing congestion episode. The smoothing is still required, however, because of drastic window cuts inherent in TCP. We used $\alpha_a = 0.2$ in the results.

3.2.3 Removing Controls

As we developed the scheme, we found that control removal is a much more challenging than setting it. The first problem is that the effects of the congestion may linger on long after the congestion source has quieted. Thus a premature return to "uncongested" state could quickly go back to congested state with incorrect parameters latched in. A more difficult problem is that shortly after the congestion source goes off, things begin to look normal until the squeezed flows are able to increase their windows to take up the slack. This brief period forms a trap for removing controls that must be avoided. Fortunately, keeping the controls on somewhat longer than necessary has no serious consequences.

The basic triggers for control removal are: (a) drop in current_rtt to the level of base_rtt, and (b) rise in avail_bw to beyond reqd_bw. The accumulated backlog during the congestion period will mean that the application will drive more than reqd_bw for some time – this creates a "hump" in the BW usage. However, this is very much a function of the application behavior. For example, if the application cannot get its IPC traffic through, it will simply slow down without any "backlog" per se. These considerations make control removal somewhat complex. Basically, the scheme includes a provision of "going over the hump" (in cases of backlog), ensuring that control removal is not done for some minimum period following normal behavior (to avoid the trap), and releasing control if no hump is observed. Also, the removal for any given connection simply puts it in "ready-to-exit" state. The actual removal of controls is done only when all connections in the group have migrated to "ready-to-exit" state.

3.3 Collective Bandwidth Control

Given the required bandwidths of all connections within a CCS (collective control set), the final question is how to make TCP connections divvy up the bandwidth properly. Our goal here was to not to perturb existing TCP implementations as far as possible so that we don't need to deal with the impact of proposed scheme on fast retransmit, fast recovery, SACK, RTT estimation, etc. Consider the connection $i \in 1..N$ in the CCS with N connections. A simple way to enforce bandwidth division is by capping the transmit window size. The cap, denoted W_i^{max}, is given by:

$$W_i^{max} = \min(\eta\lambda_a, \lambda_{ri}) RTT_{avg} \lambda_{ri} / \overline{\lambda_r} \quad (1)$$

where η is a parameter (to be discussed shortly), RTT_{avg} and $\overline{\lambda_r}$ are average RTT and required bandwidth over the CCS (e.g., $\overline{\lambda_r} = \sum_{j=1}^{N} \lambda_{rj}/N$). The intent of the equation is simply to divide up the available bandwidth in proportion of individual requirements without exceeding anybody's requirements.

The parameter η plays crucial role in the control. If $\eta \leq 1$, the controller will be unable to make use of any additional bandwidth that becomes available and thus cannot recover as the interfering traffic goes down or disappears completely. In fact, even η slightly larger than 1.0 may make the connection too sluggish to survive under competition from higher priority interfering traffic. On the other hand, a large η implies that the connection will be very aggressive in grabbing additional available bandwidth and thus lead to more severe oscillations in the thorughput. It was found that $\eta = 1.15$ is aggressive enough to work even under very severe congestion scenarios. A mathematical model characterizing the behavior with respect to η is currently being investigated.

4 Experimental Results

In order to study the behavior of proposed scheme, we first considered a rather simple simulation setup using OPNET (www.opnet.com). The choice of OPNET makes the simulation quite realistic because OPNET provides detailed implementation of all network layers including Ethernet MAC, IP, TCP, etc. Opnet also provides several built-in applications and detailed switch/router models. In fact, we have ourselves built a very comprehensive cluster database model called DCLUE [1] partly to demonstrate the need for and benefits of collective BW control. However, in this paper, we choose to concentrate on simple Opnet applications only. The next few sections describe the model and its performance under a variety of scenarios. Unfortunately, space does not permit to show all combinations of parameters that were tested.

4.1 Model Architecture

Fig. 2 shows the network model used in the experiments reported here. Server1 hosts two database applications and Server2 hosts an FTP application. Clients 1 and 2 generate queries for DB applications 1 and 2 respectively. Client 3 generates file transfer requests for Server2. All traffic goes via identically

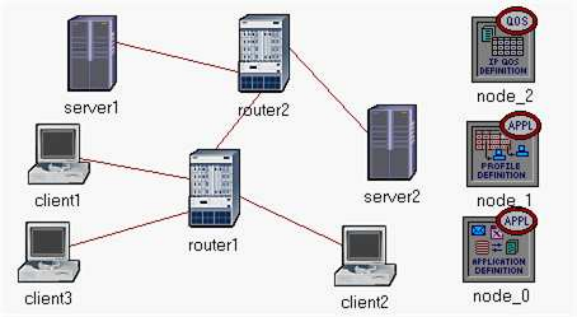

Fig 2: A network to with two database and one FTP applications

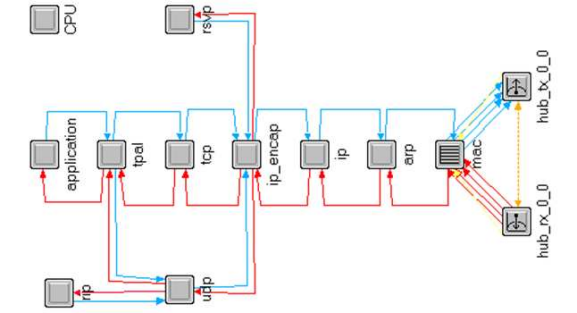

Fig 3: Node model for Servers

configured routers 1 and 2. All links shown are standard 10 Mb/sec FDX (full duplex) with default parameters. The router models try to mimic 3Com CB6000 router features. The 3 nodes on RHS in Fig. 2 are for setting QoS, application profiles, and application definitions respectively. Both the client and server application layers ride TCP/IP/MAC layer as illustrated by the node model for the server in Fig. 3.

Each database application is designed to run over a single permanent TCP connection between the client and the server. The retransmission limit for these connections was made very high so as to avoid an automatic connection reset when too many retransmissions take place. The FTP application, on the other hand, establishes a new TCP connection for each file transfer. The database traffic is 100% query type (i.e., no updates) and thus most of the traffic is from server to client direction. On the other hand, the FTP traffic is split 50-50 between gets and puts. DB traffic from both clients has exponential inter-arrival times and with response sizes uniformly distributed in the range of 8KB and 16KB. The main distinction between clients 1 and 2 is that client 1 drives twice as much response BW as client 2

(4.8 Mb/sec vs. 2.4 Mb/sec excluding about 10% application and lower layer protocol overhead). The FTP requests also have exponential inter-arrival times but involve 25 parallel streams, each requesting files uniformly distributed in the range of 5KB and 35KB. The total *offered* FTP traffic is set at 2 Mb/s, 4 Mb/s and 8 Mb/s (again, excluding about 10% protocol overhead).

It is clear that the link between the two routers is most loaded and we want to examine consequences of this. Also, the intent of the setup is to treat FTP traffic as the "interfering traffic" and examine the performance impact on the database traffic. This also means that *the autonomic monitoring and control discussed here are applied only to the DB traffic (which runs on permanent connections); there is no change to the FTP traffic.* The FTP traffic comes on at time 30 secs, stays on for 60 secs, and then goes away. The database traffic is always on (starting at time 5 secs to allow for routing table updates to settle down and TCP connections to be established).

In order to stress test the functioning of the control scheme, by default, we made the FTP traffic higher priority than DB traffic. (The "lazy setup" situation where all 3 traffic types have the same priority is also discussed later.) FTP was given higher priority by assigning it a diff-serv code (DSCP) of AF31 as opposed to AF21 to database traffic. By default, this amounts to a queue size for FTP traffic at the routers being twice as much as for the DB traffic.

We considered 2 queue sizes: (a) *full queues*, which are sized at $Q_{n1} = 120$ and $Q_{n2} = 60$ packets respectively, and (b) *half queues*, which are sized as $Q_{s1} = 60$ and $Q_{s2} = 30$ packets respectively. Note that a higher AF setting does *not* result in strict priority scheduling of FTP traffic. This is a characteristic of the used router model. The routers are configured with centralized buffering scheme (shared by all ports) instead of per port buffering. Given infrequent congestion episodes in a data center, the former is much more cost effective.

The routers were configured for RED and involve 3 parameters: (a) min threshold $L_{\min}$, (b) max threshold $L_{\max}$, and (c) drop percentage P_D. The basic idea is to drop P_D% of the packet when $L_{\min}$ threshold is exceeded, and drop all packets when $L_{\max}$ threshold is exceeded. We made $L_{\max}$ same as the queue size and $L_{\min}$ as 1/2 as much. We also chose $P_D = 10\%$. It is well known that proper setting of RED parameters is extremely difficult [10]; therefore, our settings are rather arbitrary and do not change

(except for the 2 queue sizes studied above).

Opnet's original TCP implementation supports most of the popular variants (e.g., Reno, NewReno, Tahoe), SACK, ECN, etc. We modified it with CBC so that all these features will continue to work. *Since the original version is designated as v3; therefore, we called our modified version as TCPv4.* The results below compare v3 and v4 under a variety of circumstances. It is important to note that TCPv4 comes into play only when there are competing connections at an endpoint. In the results shown, FTP traffic remains TCPv3 at all times. By default, ECN was disabled, receiver buffer was set to 64KB, and various TCP timers were reduced to correspond to data center environment. To exhibit TCPv3 vs. v4 performance, we will show the following key parameters in the results that follow:

1. Carried interfering FTP traffic (server to client direction). Under severe congestion, this traffic will be less than offered even with FTP traffic at a higher DSCP value.

2. Overall dropped traffic at router2. (Router1 to router2 traffic is small and does not experience drops).

3. RTT and retransmissions for the TCP connection carrying DB1 traffic (server to client direction). DB2 traffic parameters are generally similar and not shown.

4. Carried DB1 and DB2 traffic (server to client direction).

The simulation collects a huge number of other parameters as well (e.g., window size, in-flight size, queuing delays, etc.) but these are not shown to avoid clutter.

4.2 Full Queue Results

Figs 4-9 show comparative performance under offered FTP traffic of 8 Mb/sec (and full router queues). Note that this situation corresponds to a severe congestion at inter-router link since we are trying to drive $\tilde{1}$6.7 Mb/sec traffic through a 10 Mb/sec link.

TCPv3 behavior is shown in red lines (lighter shade in printouts) in all graphs, and is discussed in this paragraph. The severe congestion results in a significant amount of packet losses and retransmissions

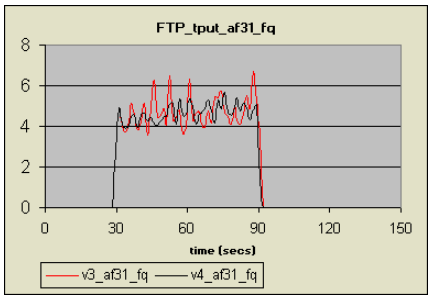

Fig 4: FTP carried traffic: AF31, FullQ, 8 Mb/s FTP traffic

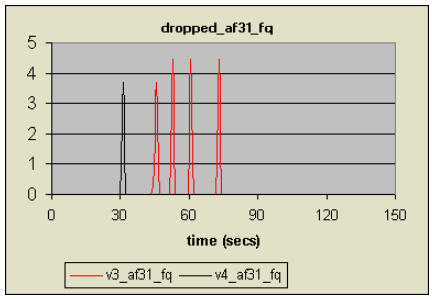

Fig 5: IP Packet Losses: AF31, FullQ, 8 Mb/s FTP traffic

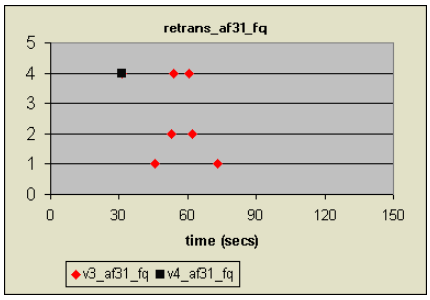

Fig 6: Retransmissions for DB1: AF31, FullQ, 8 Mb/s FTP traffic

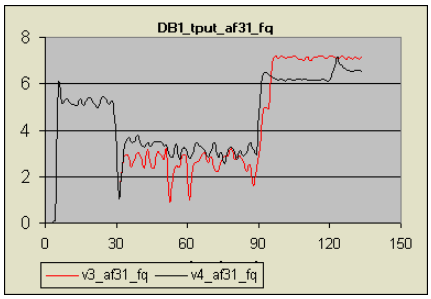

Fig 7: DB1 carried traffic: AF31, FullQ, 8 Mb/s FTP traffic

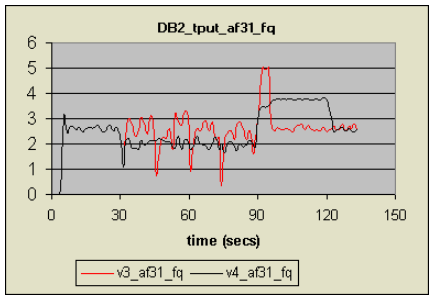

Fig 8: DB2 carried traffic: AF31, FullQ, 8 Mb/s FTP traffic

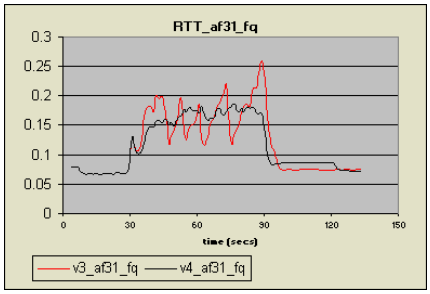

Fig 9: TCP RTT for DB1: AF31, FullQ, 8 Mb/s FTP traffic

while TCPv3 attempts to cut down the traffic. The RTT also increases due to congestion. The most interesting behavior is that of DB1 vs. DB2 throughput. It is seen that DB1 traffic is affected severely but DB2 traffic almost stays at the same level! In other words, TCP pretty much equalizes the bandwidth of the two flows even though we started with 2:1 ratio. This is, as expected, for TCP.

Let us now examine the TCPv4 behavior (shown in dotted lines). It is seen that both IP packet drops

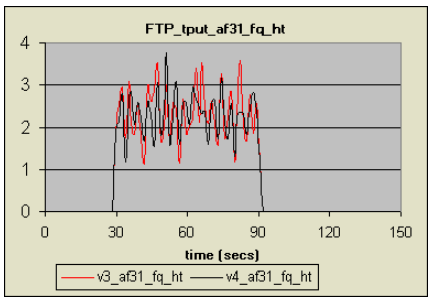

Fig 10: FTP carried traffic: AF31, FullQ, 4 Mb/s FTP traffic

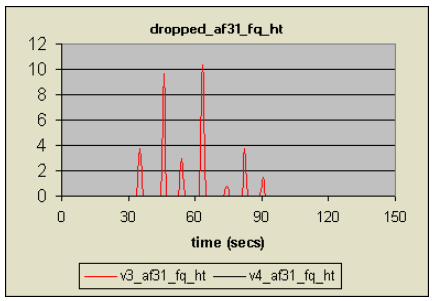

Fig 11: IP Packet Losses: AF31, FullQ, 4 Mb/s FTP traffic

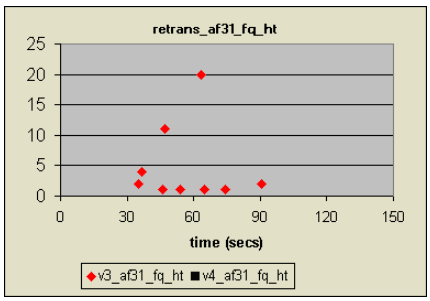

Fig 12: Retransmissions for DB1: AF31, FullQ, 4 Mb/s FTP traffic

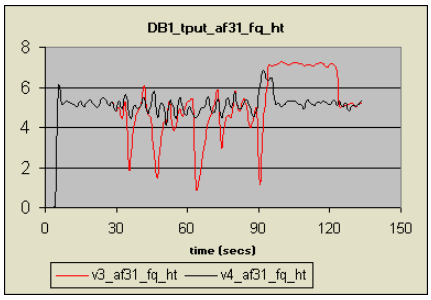

Fig 13: DB1 carried traffic: AF31, FullQ, 4 Mb/s FTP traffic

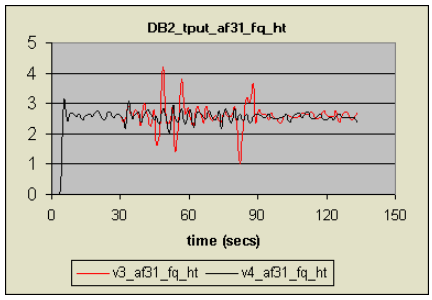

Fig 14: DB2 carried traffic: AF31, FullQ, 4 Mb/s FTP traffic

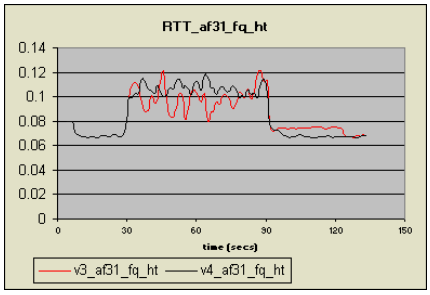

Fig 15: TCP RTT for DB1: AF31, FullQ, 4 Mb/s FTP traffic

and retransmissions go down drastically, and the RTT shows much lower fluctuations. All of these are highly desirable from a data center perspective as discussed above. It is worth remarking that with TCPv4, packet drops happen only initially which already points to a more effective congestion control. Furthermore, unlike TCPv3, the DB2 flow also suffers, in a proportion closer to 2:1 (but not quite). As a bonus, the throughput is far less variable than for TCPv3. The overall router link utilization (not shown for brevity) also shows much fewer and shorter excursions below the nominal 100% level. Even the interfering FTP traffic experiences a much smoother throughput for v4. It follows that CBC is working extremely well, although the 2:1 throughput ratio isn't quite achieved (more on this in half-queue results later).

Figs. 10–15 show the same set of graphs when the interfering FTP traffic is halved to 4 Mb/s. This is still a fairly serious case of bandwidth shortage. The first thing to note from these graphs is a significant increase in TCP oscillations, especially for TCPv3. This is to be expected as TCP gets more chances to increase its window, only to be forced to cut back. We do not address this issue of TCP in this paper since it can be adequately addressed by well-known techniques such as delay based control [6] and slowly responsive congestion control algorithms [13]. Notice that in Fig. 10, there is little difference between TCPv3 and "TCPv4" cases since both are really TCPv3 for the FTP traffic. One could use TCPv4 for the transient FTP connections as well, but analysis of such situations is beyond the scope of the paper.

It is seen from Figs. 11 and 12 that TCPv4 for DB traffic completely avoids any packet drops and retransmissions. As a result, the interfering traffic basically has no impact on DB1 and DB2 throughput for TCPv4. In contrast, TCPv3 still experiences significant losses, retransmissions and throughput irreg-

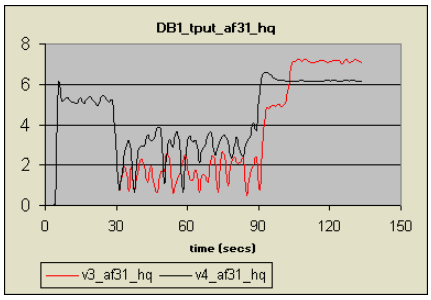

Fig 16: DB1 carried traffic: AF31, HalfQ, 8 Mb/s FTP traffic

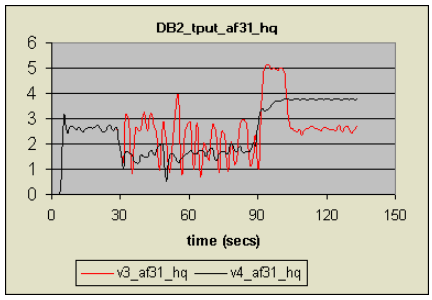

Fig 17: DB2 carried traffic: AF31, HalfQ, 8 Mb/s FTP traffic

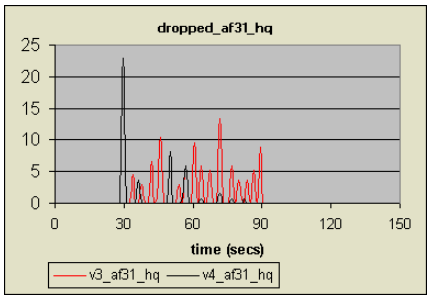

Fig 18: IP Packet Losses: AF31, HalfQ, 8 Mb/s FTP traffic

ularities. It is important to point out, however, that *the loss prevention is primarily an effect of delay related control and should be observed with delay sensitive TCP versions such as Vegas or Fast.* The relevant point to note from the perspective of this paper is that both connections continue to maintain 1:2 throughput ratio.

We repeated the runs with only 2 Mb/s interfering FTP traffic. The results are not shown for space reasons, but they were similar to 4 Mb/s case. In particular, TCPv3 still experienced losses and some sharp throughput dips (the total offered to inter-router link was slightly above 10 Mb/s). Needless to say, TCPv4 did just fine here. In fact, the only point of even doing this test was to check (a) whether the control goes in effect (it did), and (b) whether there are any problems in terms of staying on with the control during the congestion period (none), and (c) whether the control is released promptly and permanently when the congestion ends (it did).

4.3 Half Queue Results

Figs 16-18 show selected parameters with half-queue and 8 Mb/s interfering traffic. As expected, a shallower queue leads to much higher losses, but TCPv4 still shows a concentration of initial losses (when the control is taking effect) followed by much lower losses than for TCPv3. The losses do, however, result in sharp throughput dips (since we have not changed the rest of TCP). Still, the throughput is far smoother for TCPv4 than for TCPv3.

The most interesting result here is that TCPv4 does indeed cut down the two flows just enough to

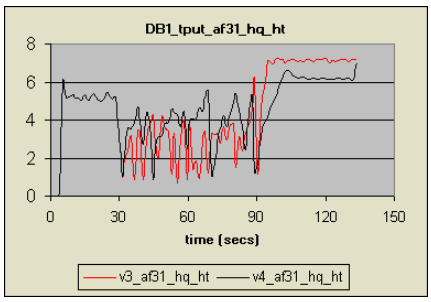

Fig 19: DB1 carried traffic: AF31, HalfQ, 4 Mb/s FTP traffic

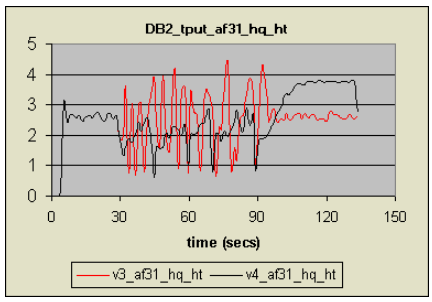

Fig 20: DB2 carried traffic: AF31, HalfQ, 4 Mb/s FTP traffic

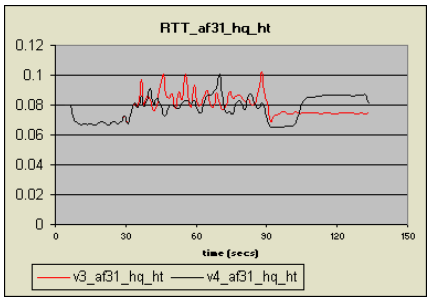

Fig 21: TCP RTT for DB1: AF31, HalfQ, 4 Mb/s FTP traffic

maintain approximately 2:1 ratio. Recall that the full queue case wasn't quite able to do that. In fact, we studied the *double queue* case as well (not shown here), and found that it primarily cuts down DB1 throughput. The conclusion is that large router queues have the effect of damping the effect of endpoint control mechanisms. Thus, there is interesting tradeoff in data centers: larger queues avoid losses but at the cost of less effective endpoint control, higher latencies, and higher cost. There is a considerable interest in data centers to limit losses and keep queues small [5]; therefore, we believe that difficulty in dealing with large router queues is perhaps not that crucial.

Figs 19-21 show selected half-queue parameters under 4 Mb/s interfering traffic. The results here can almost be predicted from the cases considered above. The choppiness of the traffic increases because of less severe overload, however, as a result of a small router queue, the DB1 to DB2 throughput is able to maintain the approximate 2:1 ratio. The losses (not shown) are still smaller for TCPv4 and concentrated initially. Consequently, the throughput is smoother for TCPv4 than for TCPv3.

It is worth focusing a bit on the behavior beyond time 90 secs when the interfering traffic goes away. It is evident from Figs 16, 17, 19, and 20, that TCPv4 recovers a bit more slowly than TCPv3 when the congestion ends. (This is related to the choice of η factor). As a result, there is a small period where the RTT falls to a normal level in TCPv4 before climbing up (see Fig 21). This is the "trap" that we talked about in the context of control removal. Also note that since the control stays on during the backlog period after the end of congestion, TCPv4 attempts to achieve the 2:1 throughput ratio even during this period. In contrast, TCPv3's behavior in this case is entirely dependent on how much backlog each connection accumulated during the congestion period.

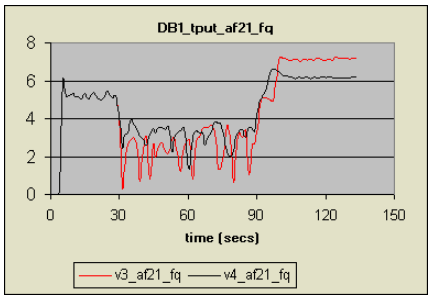

Fig 22: DB1 carried traffic: AF21, FullQ, 8 Mb/s FTP traffic

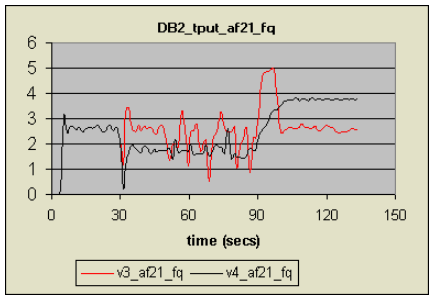

Fig 23: DB2 carried traffic: AF21, FullQ, 8 Mb/s FTP traffic

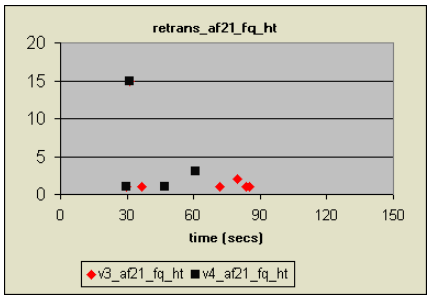

Fig 24: Retransmissions for DB1: AF21, FullQ, 8 Mb/s FTP traffic

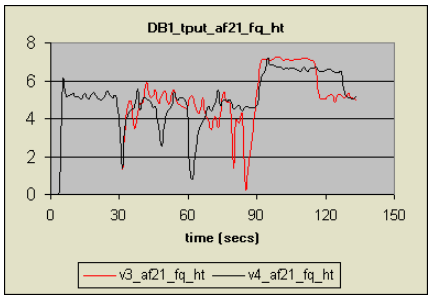

Fig 25: DB1 carried traffic: AF21, FullQ, 4 Mb/s FTP traffic

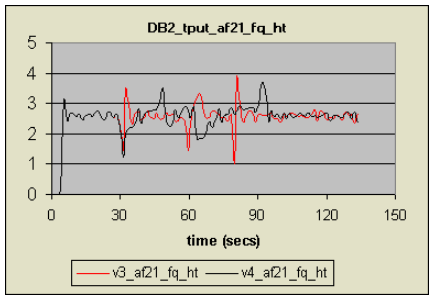

Fig 26: DB2 carried traffic: AF21, FullQ, 4 Mb/s FTP traffic

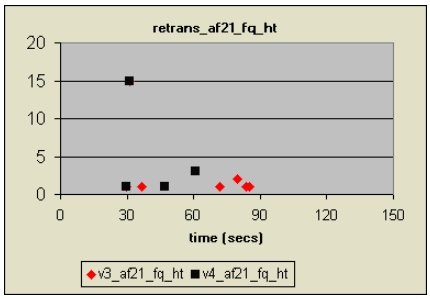

Fig 27: Retransmissions for DB1: AF21, FullQ, 4 Mb/s FTP traffic

4.4 Equal Priority Results

As stated earlier, by default we put FTP traffic at a higher priority (AF31) in order to allow it to inflict maximum damage to the DB flows of interest. In this section, we consider the situation where all 3 flows are at the same priority level. The equal priority case is perhaps more relevant when we consider non-streaming traffic within a data center, since, as remarked earlier, no priority setups will usually be in place in practice.

With equal priority, the actual DSCP choice (and corresponding queue size and other parameters) determine the overall performance. Figs 22 -24 show selected results with 8 Mb/s interfering traffic and all flows given AF21 DSCP. Although one might expect that the interfering at a lower priority will cause fewer losses/retransmissions in the DB traffic, just the opposite happens. This can be seen from Fig 6 vs. 24 and is a result of more severe packet drops at the router by the RED mechanism due to smaller overall buffer size capacity. Consequently, the throughput behavior is less stable than in section 4.2 but

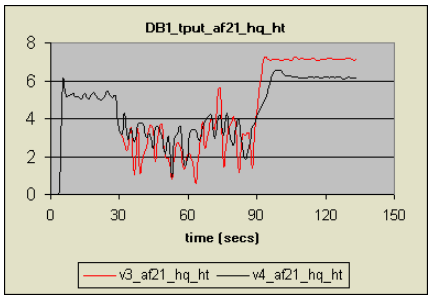

Fig 28: DB1 carried traffic: AF21, HalfQ, 4 Mb/s FTP traffic

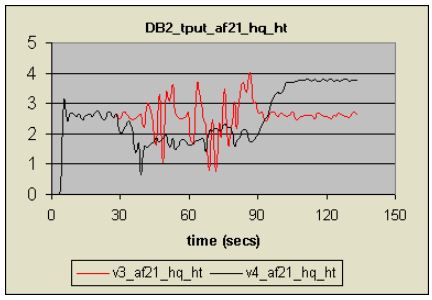

Fig 29: DB2 carried traffic: AF21, HalfQ, 4 Mb/s FTP traffic

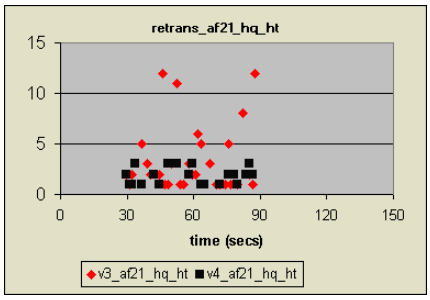

Fig 30: Retransmissions for DB1: AF21, HalfQ, 4 Mb/s FTP traffic

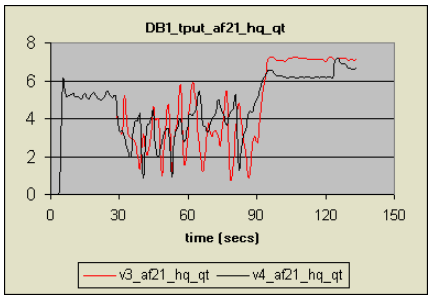

Fig 31: DB1 carried traffic: AF21, HalfQ, 2 Mb/s FTP traffic

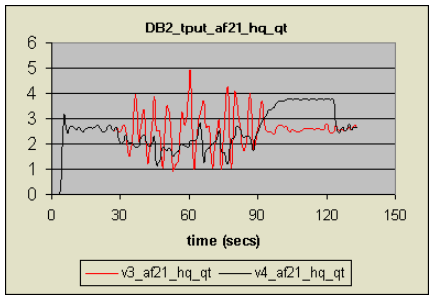

Fig 32: DB2 carried traffic: AF21, HalfQ, 2 Mb/s FTP traffic

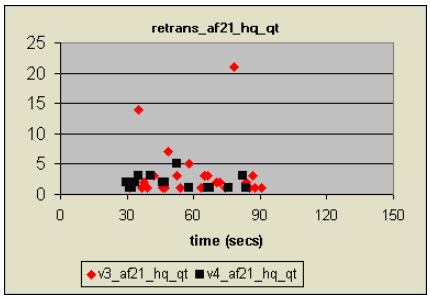

Fig 33: Retransmissions for DB1: AF21, HalfQ, 2 Mb/s FTP traffic

otherwise similar.

Figs 25 -27 show selected results with 4 Mb/s interfering traffic and all flows given AF21 DSCP. Again, the behavior is as expected: higher packet drops by RED results in less stable behavior than in section 4.2, but TCPv4 performs significantly better than TCPv3.

Finally, Figs 28 -30 and 31 -33 present equal priority results for the half-queue case, but with 4 Mb/sec and 2 Mb/sec FTP traffic respectively. As expected, the packet drops and retransmissions increase futher due to shallower queue, so much so that even the 2 Mb/sec FTP traffic is no longer loss free. In turn, the behavior becomes much less stable. Still, TCPv4 behaves much less erratically than TCPv3. Furthermore, it is able to achieve the approximate 2:1 ratio in both cases. (This is not surprising since small router queue allows endpoint to to have an effective control.)

Several other cases such as DB traffic using DSCP of AF31 and FTP using DSCP of AF11 were also used, but the space does not permit a detailed discussion. In particular, when all flows use AF31, TCPv4

is able to avoid losses completely, but TCPv3 still has a few losses.

5 Conclusions and Future Work

In this paper we studied the concept of coordinated bandwidth control in data centers and showed in a simple setting how such a scheme can benefit applications. The scheme autonomically monitors the traffic and controls related connections together in the face of congestion. Part of the further work on this topic is to show how the scheme can benefit more complex applications such as cluster databases, for which we have developed a comprehensive model [1].

Although the results in this paper indicate that the scheme is promising, we do not claim to have tied all the loose ends. Clearly, the scheme needs to be evaluated much more thoroughly with varying traffic types, overload scenarios, router settings, etc. A related question is whether the parameter values chosen in this paper can work well across the board and if not, how can we choose them autonomically? A mathematical model characterizing the dynamics of the scheme as a function of the parameter η is essential and being investigated. Determination of collective control set (CCS) is another topic that is currently being investigated along with the question of how to extend the scheme for switch congestion. Finally, extention of the scheme to properly handle non-permanent connections is still a work in progress.

References

[1] K. Kant, A. Sahoo and N. Jani, "DCLUE: A Distributed Cluster Emulator", available at http://www.ccwebhost.com/DCLUE.

[2] N. Jani and K. Kant, "SCTP performance in data center environments", available at kkant.ccwebhost.com/download.html.

[3] T. Shanley, *Infiniband Network Architecture*, Mindshare Inc., 2002.

[4] G. Regnier, S. Makineni, et. al., "TCP onloading for data center servers", Special issue of IEEE Computer on Internet data centers, Nov 2004 (Eds. K. Kant & P. Mohapatra).

[5] G. Mcalpine, M. Wadekar, et. Al., "An architecture for congestion management in Ethernet clusters", submitted for publication.

[6] C. Jin, D. X. Wei, and S. Low, "Fast TCP: Motivation, Architecture, Algorithms, Performance", Proc. of Infocom 2004.

[7] L.S. Brakmo and L. Peterson, "TCP vegas: end-to-end congestion avoidance on a global Internet", IEEE J-SAC, vol 13, no 8, pp 1465-80, Oct 1995.

[8] S. Jin, L. Guo, et. al., "A spectrum of TCP friendly window based congestion control algorithms", IEEE/ACM trans on networking, vol 11, no 3, June 2003.

[9] J. Martin, A. Nilsson and I. Rhee, "Delay based congestion avoidance in TCP", IEEE/ACM trans on networking, vol 11, no 3, pp356-369, June 2003.

[10] S.H. Low, F. Paganini, et. al., "Linear stability of TCP/RED and a scalable control", Computer Networks, vol 43, no 5, pp633-647, 2003.

[11] F.P. Kelly, A. Mullo and D. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability", Journal of OR society, vol 49, No 3, pp237-252, 1998.

[12] J. Crowcroft and P. Oechslin, "Differentiated end to end internet services using a weighted proportional fair sharing", ACM Sigcomm, vol 28, no 3, July 1998.

[13] D. Bansal, H. Balakrishna, et. al., "Dynamic behavior of slowly responsive congestion control algorithms", Proc. of SIGCOMM 2001.