Typical versus Worst Case Design in Networking

Nandita Dukkipati, Yashar Ganjali, Rui Zhang-Shen
Computer Systems Laboratory, Stanford University
Stanford CA 94305-9030
{nanditad, yganjali, rzhang}@stanford.edu

Abstract— **Networking research has a strong inclination toward designing systems for the worst case scenarios. On the face of it, this seems reasonable – designing for the worst case gives strong guarantees on the system. However, this often comes with a sacrifice for the typical case performance. Drawing from our experience, this paper argues and gives evidence of the enormous benefits of typical case design in performance, cost, and complexity. We use examples to illustrate that it is possible to design for the typical case without overly hurting the worst case: sizing router buffers for the typical case means orders of magnitude reduction in buffer size, lower network delay and jitter, and a simplified router architecture, all without sacrificing link utilization; a congestion control algorithm designed for the typical case can finish flows several orders of magnitude faster than the existing and the newly proposed algorithms.**

Index Terms— **Typical case, Worst case, Congestion Control, Buffer sizing, Network design, Load Balancing**

I. Introduction

This paper is motivated by the observation that Networking research gravitates toward designing and optimizing systems for the worst case. Some examples are: packet classification algorithms [1], packet buffer architectures for switches [2], and congestion control protocols [3]. This is contrary to the "hint" well known in designing computer systems, formalized by Lampson [4]: "Handle normal and worst cases separately as a rule, because the requirements for the two are quite different. The normal case must be fast. The worst case must make some progress." The hint has been applied well in computer systems design, but not as much in Networking. In this paper we argue and give evidence of the practicality of this hint in Networking.

The strong preference for worst case design in Networking as opposed to designing for the normal case is primarily due to the following:

1) Often, the typical case is only known after systems are deployed and extensive measurement and analysis infrastructure is in place. Even when data is available it is often proprietary and not easily accessible, such as the traffic matrices in a backbone network [5].
2) It is hard to quantify and verify performance for the typical case. Statistical analysis helps to an extent but is often too complicated to extend to real complex systems. On the other hand, it is easier to quantify and verify the worst case performance, for example, requiring 100% throughput or zero loss for any traffic pattern.
3) Designing for worst case feels good and safe. It is widely believed that, (a) if you design something that works well for worst case scenarios it must work even better for typical case, and (b) a system designed to work well under typical case will fail catastrophically under a worst case. Both of these, we will argue, are myths.

Given the difficulty of designing for the typical case, why not just design networking systems for the worst case? Some reasons not to do so are:

1) It often hurts the typical case in terms of performance, resources needed, and complexity by a factor large enough to be concerned about. For example, sizing buffers in backbone routers for worst case requires orders of magnitude more buffering than typical case, congestion control designed for no losses in the worst case hurts flow performance by two orders of magnitude in the typical case.
2) The worst case a system is designed for is often the theoretical worst case and not necessarily one that could happen in practice. For example, router buffers today are sized for the case where flows are fully synchronized, but in a backbone router where thousands of flows coexist, this can hardly happen. Optimizing for such a worst case seems unwarranted.
3) It is possible to design a system which is optimized for the typical case and can handle the worst case gracefully.

We will take the following position in this paper: *Design for the common case, unless designing for the worst-case is either absolutely necessary or cheap*. We draw attention to our experience that there can be enormous benefits in designing for the typical case, sometimes by several orders of magnitude in typical case performance, cost, and complexity, while continuing to function under the worst case. We will use three examples to drive home this point. All three examples relate to network congestion and the worst and typical cases refer to the offered load:

1) *Buffer sizing in core routers*: Today the core router buffers are sized for the worst case [6] – small number of flows or synchronized flows. The typical case in the core is desynchronized statistical mix of flows. It turns out that buffer sizes smaller by orders of magnitude suffice for the typical case [7] [8].

2) *Congestion control*: TCP is conservative in assigning bandwidth to flows; new protocols like eXplicit Control Protocol (XCP) [3] are even more conservative and are optimized for worst case traffic patterns. In the typical case of a statistical mix of flow sizes and random arrivals, Rate Control Protocol (RCP) [9] – a congestion control protocol designed to work well in the typical case – can complete flows close to the

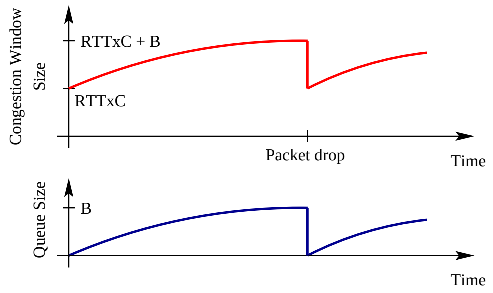


Fig. 1. Evolution of TCP's congestion window size over time. The source node sees a packet drop when both the pipe and the buffer are full, *i.e.*, when we have $\overline{\text{RTT}} \times C + B$ outstanding packets. The congestion window size must remain above $\overline{\text{RTT}} \times C$ to keep the link 100% utilized.

minimum achievable time which is several orders of magnitude shorter than TCP and XCP.

3) *Backbone network design*: Worst case design is warranted for if it is either absolutely necessary, or is cheap enough, as it is in the case of backbone network design. The newly proposed Valiant Load-balancing (VLB) [10] in backbone networks can guarantee 100% throughput for all allowable traffic matrices, while requiring no extra cost and having little or no effect on the average case performance.

II. Buffer Sizing in Routers

An example illustrating the immense benefits of designing for the typical case is router buffer sizing. How much buffering do Internet routers need so as to guarantee near 100% link utilization? A universally applied rule-of-thumb states that each link needs to have a buffer size of $\overline{\text{RTT}} \times C$, where $\overline{\text{RTT}}$ is the average two way propagation delay (round-trip time) of the packets going through the router, and C is the data rate of the link. As an example, if the average round-trip time of the packets is 250 ms, and the capacity of the core link is 40 Gb/s, the buffer size needed is 10 Gb, which is about 1,250,000 packets (assuming the average packet size is 1 KB).

Interestingly, this widely used rule-of-thumb comes from a setting with a *single* TCP flow going through the core router. TCP is a window based congestion control scheme. A TCP source can change its packet injection rate by varying the size of the congestion window. The maximum link utilization is achieved using the well-known additive increase multiplicative decrease (AIMD) scheme. In order to keep the core link 100% utilized, we need to have a buffer big enough so that it can keep the link busy after the sender halves the congestion window. Figure 1 depicts the congestion window size and the buffer occupancy of a single TCP flow. If the buffer size is B, the first packet drop occurs when the congestion window size is $\overline{\text{RTT}} \times C + B$. The congestion window size must remain above $\overline{\text{RTT}} \times C$ at all times in order to keep the link 100% utilized, which gives us the famous $B = \overline{\text{RTT}} \times C$ rule-of-thumb.

Now, let us consider a network with more than one TCP flow. Again, each of these flows uses TCP's AIMD rule to change its window size. As we can see in Figure 2(a), if all of the flows are completely synchronized, the variations in the sum of congestion window sizes are similar to that of a single-flow, and therefore we will need the same amount of buffering as indicated by the original rule-of-thumb. However, if for any reason the flows become desynchronized, variations in the sum of congestion window sizes are significantly reduced (Figure 2(b)). The amount of buffering must be large enough to accommodate the variations in the sum of congestion window sizes. Therefore, by reducing the variations, we expect that a smaller buffer size will still lead to 100% link utilization.

Are TCP flows going through the core routers synchronized or not? Using theory, simulations, and experiments with real routers, Appenzeller *et al.* showed that as we increase the number of flows in the core of the Internet, they become more and more desynchronized [7]. More precisely, they showed that if the number of flows are increased from one to N, due to the statistical multiplexing of flows, the size of the variations is reduced by a factor of $1/\sqrt{N}$ [7]. Therefore, the buffer size needed in this case becomes $\overline{\text{RTT}} \times C/\sqrt{N}$. In the example we mentioned before, if there are 10,000 flows in the core link, the buffer size will be reduced to 12,500 packets from the 1,250,000 packets suggested by the original rule-of-thumb; this is a two orders of magnitude reduction in the buffer size.

Let us step back for a second and take a closer look at these results. The original rule-of-thumb is based on the assumption that there is only one flow through the bottleneck link. Although this worst case scenario can happen, its probability is extremely small especially in the core of the Internet. Typically, there are thousands of flows going through each core link and the result of Appenzeller *et al.* takes advantage of this to significantly reduce the buffer size in core routers. We note that this reduction in buffer size doesn't cost anything in terms of system performance, *i.e.*, the link remains 100% utilized. At the same time, the consequences of small buffers are: (a) reduced delay, (b) reduced jitter, and (c) simplified router design. The last point comes from the fact that the memory needed for buffering packets can now be placed on chip, thus eliminating complicated off-chip memories and the transfer of packets back and forth between the buffer and the processing unit of the router.

Another assumption embedded in the original rule-of-thumb is that the core link should be 100% utilized. This is another worst case scenario assumption since the Internet backbone is highly over-provisioned today, and typically backbone links work at very low utilizations [11]. Now the question is, if we relax the assumption of the link being always 100% utilized, can we reduce the buffer sizes even more?

Enachescu *et al.* [8] showed that the buffer sizes can be even further reduced if the links are not required to be 100% utilized all the time. In fact, with a buffer size of just 10-20 packets (which is about five orders of magnitude smaller than the original rule-of-thumb) we can guarantee a link utilization of at least 75%. Of course if the load of the system is low (as it is in today's Internet) we will not see any degradation in system performance. However, if we keep increasing the load we might not be able to get 100% link utilization. In other words, we will be able to reduce the buffer size to 10-20 packets, at the cost of losing (at most) 25% of link utilization.

The value of this trade-off becomes clearer if we consider the consequences of this result: 10-20 packets is exactly the amount of all-optical buffering promised by recent advances

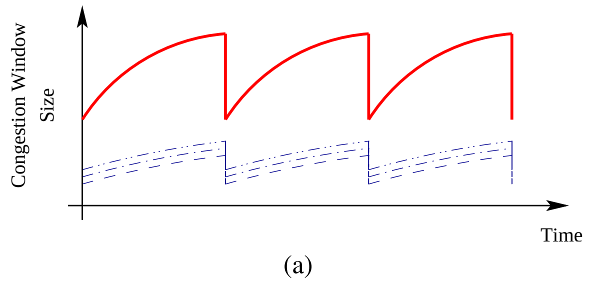
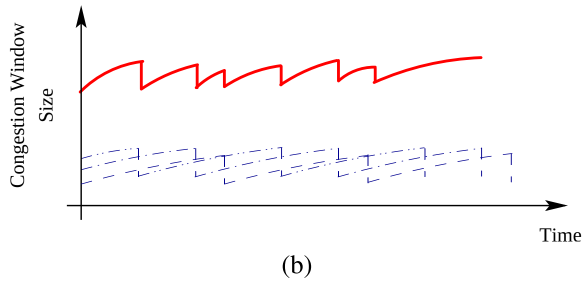


Fig. 2. Dashed curves show the congestion window sizes of individual flows, and the solid curve depicts the sum of the congestion window sizes. (a) synchronized flows, (b) desynchronized flows.



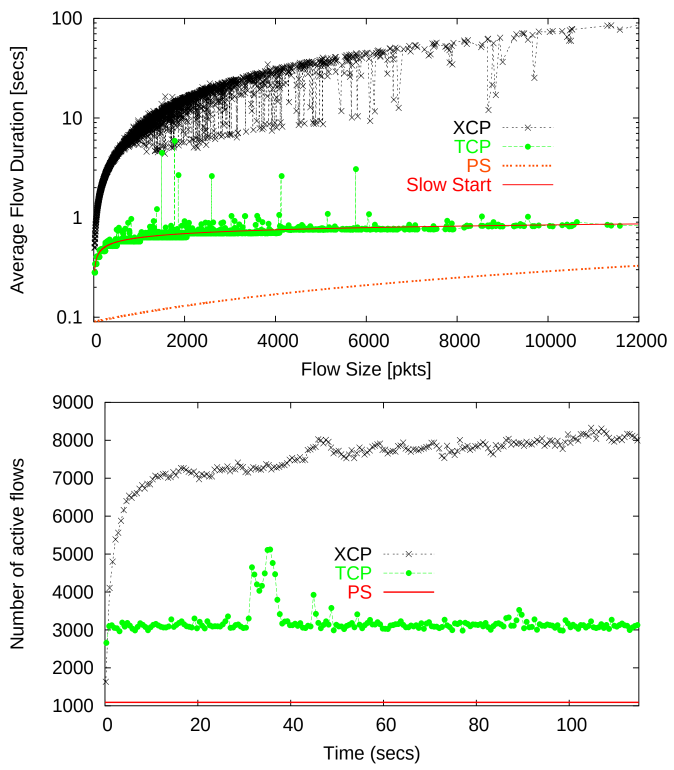


Fig. 3. The top plot shows the average flow duration versus flow size under TCP and XCP from a simulation with Poisson flow arrivals, flow sizes are Pareto distributed with mean = 50 pkts (1000 byte/pkt) and shape = 1.3, link-capacity = 2.4 Gbps, round-trip time = 60 ms, offered load = 0.9. The bottom plot shows the number of active flows versus time. In both plots the PS values are computed from analytical expressions.

in the optical buffering technologies. At the cost of losing a fixed portion of the link capacities, we can build a network of all-optical routers and guarantee their performance. The advantages of having an all-optical network include very high capacity, low power consumption, higher security, and less delay and jitter. We note that since the link capacities are extremely high in an all-optical network, the core will not be the bottleneck in the network, and losing 25% link utilization is not considered a major problem.

III. Congestion Control

TCP's congestion control is deliberately designed to be conservative so as to avoid congestion collapse. While this works very well in avoiding severe congestion, it also has the effect of making flows last many times longer than necessary in the typical case. Simple experiments indicate that with typical Internet flow sizes and offered loads, TCP does not come close to emulating the ideal processor sharing. We are comparing it to processor sharing (PS) because this is what TCP is attempting to achieve in the long term – to share the link bandwidth efficiently and fairly[1]. For example, Figure 3 compares TCP with ideal PS (as well as XCP, which we will discuss shortly). The top plot compares the mean duration of flows (how long they take to complete) as a function of flow size. The values for PS are derived analytically, and show that flows would complete an order of magnitude faster than for TCP.

There are several reasons for the long flow durations in TCP. First is slow-start, which takes several round-trip times (RTTs) to find the fair-share rate. In many cases, the flow has finished before TCP has found the correct rate. Second is the AIMD mode in which TCP adapts slowly because of additive increase. Finally, TCP fills up any amount of buffering available at the bottleneck link; extra buffers mean extra delay, which add to the duration of a flow.

Our plots also show eXplicit Control Protocol (XCP) [3]. XCP is designed to work well in networks with large bandwidth-delay product. The routers provide feedback, in terms of incremental window changes, to the sources over multiple round-trip times, which works well when all flows are long-lived. But as our simulations show, in a dynamic environment XCP can increase the duration of each flow even further relative to ideal PS, and so there are more flows in progress at any instant. XCP is slow in giving bandwidth to the flows, especially to newly starting flows. It takes multiple RTTs for most flows to reach their fair share rate (which is changing as new flows arrive). Many flows complete before they reach their fair share rate. In general, XCP stretches the flows over multiple RTTs, to avoid over-subscribing the link, and so keep buffer occupancy low.

In summary, TCP and XCP are designed for the worst case in the sense that: (a) They start flows slowly; while this is good for flash crowd like scenarios, in the typical case it makes the flows last many more RTTs longer than need-be. (b) Both give small incremental window feedback to flows in every

[1]If all flows are long-lived, TCP achieves proportional sharing; it approximately shares bandwidth as $\frac{K}{\text{RTT}\sqrt{p}}$. This reduces to processor sharing [12] if flows have the same RTT.

RTT, which works very well when most or all flows last long enough to reach their equilibrium fair share rate. However, in the typical case when most flows are capable of finishing within a few RTTs, it makes flow completion times (FCT)[2] one or two orders of magnitude higher than necessary.

A. *A simple congestion control algorithm to emulate processor sharing*

In the following we will describe the Rate Control Protocol (RCP) [9], which is a simple congestion control algorithm with the goal of finishing flows quickly – close to ideal processor sharing – irrespective of flow size distributions and network conditions. Instead of starting flows slowly or giving small incremental window feedback in every RTT (like in TCP and XCP), RCP is interested in the following question: If the router were to give a *single* rate, $R(t)$, to all flows at a bottleneck link so as to emulate PS, what should this rate be? RCP is an adaptive algorithm at the router that updates $R(t)$ to approximate PS in the presence of feedback delay without any knowledge of the number of ongoing flows, and requires no per-flow state or per-packet computation. A source transmits at the most bottlenecked rate along its path[3]. $R(t)$ is updated by the routers based on the equation below:

$$R(t) = R(t-T)\left(1 + \frac{\frac{T}{d_0}(\alpha(C-y(t)) - \beta\frac{q(t)}{d_0})}{C}\right) \quad (1)$$

where d_0 is a moving average of the RTT measured across all flows, T is the rate update interval (i.e., how often $R(t)$ is updated) and is less than or equals d_0, $R(t-T)$ is the last updated rate, C is the link capacity, $y(t)$ is the measured input traffic rate during the last update interval, $q(t)$ is the instantaneous queue size, and α, β are parameters chosen for stability and performance. The basic idea is: If there is spare capacity available (i.e., $C - y(t) > 0$), then share it equally among all flows. On the other hand if there is a queue building up ($q(t) > 0$), then the link is oversubscribed and the flow rate is decreased evenly. The expression $\frac{T}{d_0}(\alpha(C-y(t)) - \beta\frac{q(t)}{d_0})$ is the desired aggregate change in traffic in the next control interval, and dividing this expression by the number of ongoing flows, gives the change in traffic rate needed per flow. Built into Equation (1) is the router's estimate of the number of flows, $N(t)$, as $\hat{N}(t) = \frac{C}{R(t-T)}$.

B. *Example*

Figure 4 shows an example illustrative of the many simulations done with RCP [9]. The figure shows the Average Flow Completion Time (AFCT) versus flow size for RCP, TCP, XCP, and PS when flows arrive in a Poisson process and have Pareto distributed flow sizes. The AFCT of RCP is close to that of PS

[2]Flow completion time is defined as the time interval between the sender sending a SYN packet and the receiver receiving the last packet of the flow.

[3]Every packet header carries a rate field, R_p. When transmitted by the source, $R_p = \infty$. When a router receives a packet, if $R(t)$ at the router is smaller than R_p, then $R_p \leftarrow R(t)$; otherwise it is unchanged. The destination copies R_p into the acknowledgment packet, so as to notify the source. The source transmits at rate R_p, which corresponds to the smallest offered rate along the path.

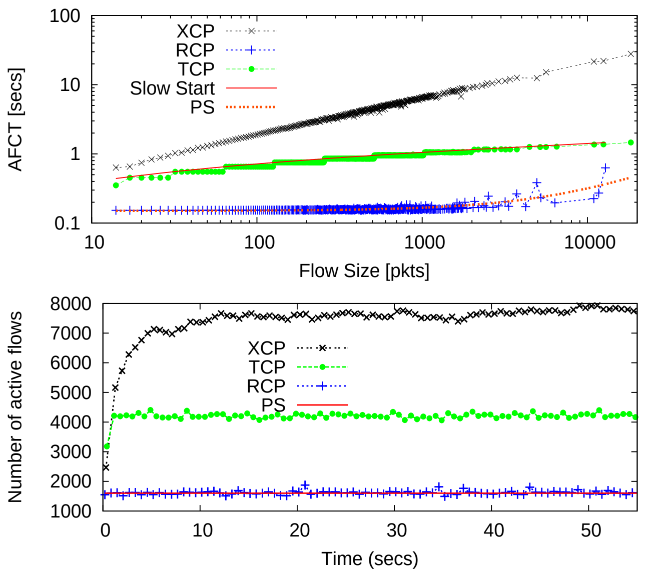


Fig. 4. Average Flow Completion Time (AFCT) for different flow sizes when $C = 2.4$ Gb/s, RTPD= 0.1 s, and $\rho = 0.8$. Flows are Pareto distributed with $E[L] = 30$ pkts, shape = 1.8. The top plot shows the flow completion times; the bottom plot shows the number of active flows over time.

and is always lower than that of XCP and TCP. TCP's delay is 4 times higher and XCP's is more than 8 times higher.

When there are only long-lived flows RCP achieves PS at a single node and max-min fairness in multiple node case, just like XCP and TCP. The worst case scenario in RCP would be a sudden large influx of new flows within one RTT. Because all flows receive a high starting rate, packet drops are more likely than in TCP and XCP. The good news is after experiencing losses in one RTT, RCP reduces $R(t)$ aggressively and the system recovers to full utilization within a few round trip times.

IV. More Examples of Typical Case Design

A. *Enforcing fairness with low implementation complexity*

Approximate Fair Dropping (AFD) [13] is a router mechanism to enforce fairness among flows while using a FIFO queue. It probabilistically drops packets on arrival based on recent queue measurements and history of the flows. AFD uses history to estimate the flow's current sending rate, and drops packets with a probability designed to limit the flow's throughput to the fair share rate. If r_i is the sending rate of flow i and r_{fair} is the fair share rate given by the solution to *link-capacity* $= \sum_i \min(r_i, r_{fair})$, then the dropping probability for flow i is $d_i = (1 - \frac{r_{fair}}{r_i})^+$. The flow throughput is capped by the fair share, $r_i(1 - d_i) = \min(r_i, r_{fair})$.

An important goal of AFD is to maintain low state and implementation complexity. Clearly, if one has to accurately track every flow's rate, it requires state which grows linearly with the number of flows. AFD does away with this by observing that it only needs to keep enough state to estimate rates which are close to or larger than the fair share rate. To estimate flow rates, AFD keeps a *shadow buffer*, b, of recent packet arrivals. When a packet arrives, it is probabilistically sampled and copied into the shadow buffer, while another packet is deleted at random from the buffer.

The tradeoff involved in AFD is as follows: while a large b will ensure better fairness properties, it also means more state and higher complexity. [13] observes from packet traces that typically most flows are slow (in bits/sec), called *turtles*, while most bytes are from a few fast flows, called *cheetahs*. It is precisely this regime (when the distribution of flow-rates is heavy-tailed) that AFD operates the best as it needs state only for the few manageable number of fast flows. In the worst case of a large number of fast or unresponsive flows, the small amount of state may not be sufficient to enforce fairness.

B. Fast packet classification

Most work on packet classification uses worst case time and the memory needed as metrics for evaluation. [14] observes that for applications not having real-time constraints, the worst case is not as important as the average case. They analyzed Tier 1 ISP data and observed a Zip-f like distribution in the usage of rules in a classifier, where a very small number of rules resolve the bulk of traffic and most rules are essentially never used. Inspired by this, they proposed traffic-aware classifiers – *traffic-aware common branch decision trees* – that have superior average-case performance while having a bounded worst-case. The worst-case for these classifiers is worse than those explicitly optimized for the worst case.

C. Web caching

Web caching systems [15] result in significant improvements in (a) bandwidth savings, (b) load balancing across servers, (c) network latency reduction, and (d) content availability. These gains are mostly in the typical case when there is temporal locality among web objects; in the worst case the original content provider will have to serve the client requests.

V. When Worst-case Design Is Warranted: Backbone Network Design

There are two situations when we would want to design for the worst case:

1) Worst-case service guarantees are required.
2) It is cheap to design for the worst case and the scheme does not hurt typical case performance significantly.

We will use the example of backbone network design to illustrate how we can design for the worst case that is both cheap and does not hurt the typical case. We will first argue why the current way of backbone design is not adequate, and then show that Valiant Load-balancing [10], a new backbone architecture, can provide worst-case guarantees at low cost.

In Internet backbone design, the common practice today is to assume some traffic matrix based on measurements and predictions, and design the network to support the traffic while adding some margins by over-provisioning. There are three problems with this approach. First, it is hard to measure and impossible to predict the Internet traffic matrix and so the design input cannot be accurate. Second, the network cannot guarantee to sustain traffic that is different from the design input; in fact, it is almost certain that some traffic that is deviant from the typical case will cause congestion. Finally, the distribution of the traffic matrix is unknown, so no prediction is available as to how often congestion happens.

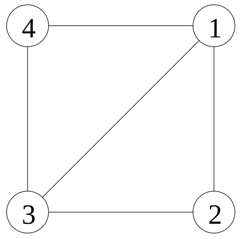


Fig. 5. Example for calculating the percentage of traffic matrices which can be supported.

To quantify the flexibility of a backbone network in terms of the set of traffic matrices that can be supported, we conduct the following study. Consider a network of N nodes. Define the traffic matrix to be $\Lambda = \{\lambda_{ij}\}$, where λ_{ij} is the traffic rate from node i to node j.[4] A node can initiate traffic to any node, so the traffic matrix is only constrained by the node capacities: node i can initiate traffic to (receive traffic from) the other nodes at a rate of no more than R_i.

So the set of allowable traffic matrices is:

$$\mathcal{T} = \{\Lambda | \sum_j \lambda_{ij} \leq R_i, \forall i; \quad \sum_j \lambda_{ji} \leq R_i, \forall i; \\ \lambda_{ij} \geq 0, \forall i,j; \quad \lambda_{ii} = 0, \forall i\}. \tag{2}$$

The $N(N-1)$ non-zero elements of the traffic matrix satisfying (2) form a convex polytope[5], the volume of which can be calculated.

Given a network, and assuming some deterministic routing scheme such as shortest path first, an additional set of constraints are introduced: $Al \leq c$, where A is the routing matrix, l is a vector consisting of λ_{ij}, and c represents the link capacities. Now the traffic matrices that can be supported by the network form a smaller polytope:

$$\mathcal{S} = \{\Lambda | \sum_j \lambda_{ij} \leq R_i, \forall i; \quad \sum_j \lambda_{ji} \leq R_i, \forall i; \\ \lambda_{ij} \geq 0, \forall i,j; \quad \lambda_{ii} = 0, \forall i; \quad Al \leq c\}. \tag{3}$$

Comparing the volume of the two polytopes $\mathcal{S}$ and $\mathcal{T}$, we can find the percentage of the allowable traffic matrices that can be supported by the network.

Figure 5 shows a small example. We assume a simple case where all nodes have the same capacity, and the network is designed for the uniform traffic. Without over-provisioning, the network can support **0.20%** of the allowable traffic matrices; with 25% over-provisioning, it can support **2.59%**; and with 50% over-provisioning, it can support **15.09%**. This example shows that the portion of traffic matrices that can be supported by the network is rather small, even when there is over-provisioning.

[4]Since the traffic destined to a node itself does not need to traverse the network, we can always set the diagonal entries of Λ to be zero.

[5]Mathematically, a *polytope* is a bounded intersection of a finite set of half-spaces.

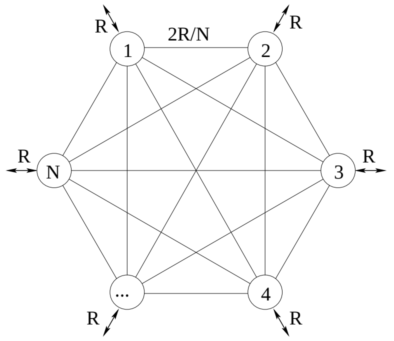


Fig. 6. Example network for Valiant load-balancing.

The Valiant load-balancing (VLB) scheme [10] is designed to support all traffic matrices (the set $\mathcal{T}$). We use the network in Figure 6, which consists of N nodes of the same capacity R, to explain the VLB scheme. A full mesh of logical links of capacity $\frac{2R}{N}$ interconnects the N nodes. Routing is done in two stages. In the first stage, each node uniformly load-balances its incoming traffic to all nodes. This means, each flow (defined by source and destination nodes) is spread evenly to N nodes. Since the incoming traffic at each node is at most R, this stage requires at most $\frac{R}{N}$ capacity on each logical link. In the second stage, the traffic is delivered to final destinations. This is the dual of the first stage. Since each node can receive traffic at the maximum rate of R, and each flow has been evenly spread N ways, again a capacity of $\frac{R}{N}$ on each link is sufficient. Thus the network with capacity $\frac{2R}{N}$ on each link can guarantee service to all valid traffic matrices.

The VLB architecture is proved to be the most efficient in terms of link capacity amongst all networks which can support all traffic matrices [16], but it may seem that the VLB network guarantees worst case performance at a cost of higher link capacities: A uniform traffic matrix would require only $\frac{R}{N}$ on every link. Our empirical study shows, however, that the cost of a VLB network is almost the same as that of a conventional network for the same traffic matrix (while the VLB network can support all traffic matrices). The reason for the similarity in cost is that in a VLB network route lookup is done only once per packet, saving cost on routers to compensate the extra expense for link capacities.

VLB guarantees worst case performance cheaply, and does not hurt the typical case performance. The load-balancing scheme makes almost all packets traverse the network twice, resulting in a increased delay. But there is no need to load-balance when the traffic is uniform. When the load becomes more and more skewed, the amount of required load-balancing increases, and can be calculated adaptively [17]. This means, the VLB scheme does not need to affect the typical case performance.

In summary, VLB is an example of an architecture which can guarantee worst case performance with little or no effect on the typical case and does not much cost more.

VI. Conclusion

All too often, networking research gets consumed by worst case design. This works very well either when it has little or no effect on the typical case performance or when the worst case scenario creates such a pathological behavior that it is worth sacrificing the typical case. As we have seen in the examples, blindly designing for the worst case could mean enormous sacrifices (in performance, resources, or complexity) in the typical case. It is the premise of this paper that we should design networking systems and infrastructure for the typical case unless the worst case is absolutely necessary or is cheap. This in turn means we need better measurements and models to characterize the nature and frequency of the worst and typical cases in the network.

VII. Acknowledgment

We thank Nick McKeown for his insightful comments. This research is supported by Stanford Graduate Fellowship, by NSF under ITR award ANI-0331653, and by DARPA/MTO under DOD-N award no. W911NF-04-0001/KK4118 and Grant no. W911NF-05-1-0224.

References

[1] P. Gupta, N. McKeown, "Dynamic Algorithms with Worst-case Performance for Packet Classification," in *Proceedings IFIP Networking*, Paris, France, May 2000.

[2] S. Iyer, R. R. Kompella, N. McKeown, "Analysis of a Memory Architecture for Fast Packet Buffers," in *IEEE High Performance Switching and Routing*, Dallas, Texas, May 2001.

[3] D. Katabi, M. Handley, C. Rohrs, "Internet Congestion Control for High Bandwidth-Delay Product Networks," in *Proceedings of ACM Sigcomm 2002*, Pittsburgh, August, 2002.

[4] B. W. Lampson, "Hints for Computer System Design," in *Proceedings of the ninth ACM Symposium on Operating Systems Principles*, New Hampshire, 1983.

[5] R. Zhang-Shen, *Private communication with Internet service providers.*

[6] C. Villamizar, C. Song, "High Performance TCP in ANSNET," in ACM SIGCOMM Computer Communication Review, Volume 24, Issue 5, October 1994.

[7] G. Appenzeller, I. Keslassy, N. McKeown, "Sizing Router Buffers," in *ACM SIGCOMM 2004*, Portland, August 2004.

[8] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, T. Roughgarden, "Routers with Very Small Buffers," in *ACM SIGCOMM Computer Communication Review*, Volume 35 , Issue 3, July 2005.

[9] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, N. McKeown, "Processor Sharing Flows in the Internet," in *Thirteenth International Workshop on Quality of Service (IWQoS)*, Passau, Germany, June 2005.

[10] R. Zhang-Shen, N. McKeown, "Designing a Predictable Internet Backbone Network," in *HotNets III*, San Diego, November 2004.

[11] S. Iyer, S. Bhattacharyya, N. Taft, N. McKeown, C. Diot, "An approach to alleviate link overload as observed on an IP backbone," in *IEEE Infocom*, San Francisco, March 2003.

[12] S. B. Fredj, T. Bonald, A. Proutiere, G. Regnie, J. W. Roberts, "Statistical Bandwidth Sharing: A Study of Congestion at Flow Level," in *Proceedings of ACM Sigcomm 2001*, San Diego, August 2001.

[13] R. Pan, L. Breslau, B. Prabhakar, S. Shenker, "Approximate Fair Dropping through Differential Dropping," in *ACM SIGCOMM Computer Communication Review*, Volume 33, Issue 2, April 2003.

[14] E. Cohen, and C. Lund, "Packet Classification in Large ISPs: Design and Evaluation of Decision Tree Classifiers," in *ACM SIGMETRICS Performance Evaluation Review*, Volume 33 , Issue 1, June 2005.

[15] G. Barish, K. Obraczka, "World Wide Web Caching: Trends and Techniques," In *IEEE Communications Magazine Internet Technology Series*, Volume 38, Issue 5, Pages 178-184, May 2000.

[16] I. Keslassy, C.-S. Chang, N. McKeown, D.-S. Lee, "Optimal load-balancing," in *Proceedings of IEEE Infocom 2005*, March 2005.

[17] H. Liu, R. Zhang-Shen, "On Direct Routing in the Valiant Load-Balancing Architecture," *IEEE Globecom 2005*, St. Louis, MO, November 2005.