

Solving Hot Spot Contention Using InfiniBand Architecture Congestion Control

Invited Paper

G. Pfister, M. Gusat, W. Denzel, D. Craddock, N. Ni, W. Rooney,
T. Engbersen, R. Luijten, R. Krishnamurthy and J. Duato

Abstract: Since at least 1985[1] it has been known that certain traffic patterns in multistage interconnection networks, hot spots, can cause catastrophic congestion and loss of throughput. No practical technique has, until now, been demonstrated to alleviate this problem, which becomes increasingly severe as network size increases and networks are driven closer to saturation. The congestion control architecture (CCA) proposed for the InfiniBand™ Architecture was alleged to be a solution, but when it was defined it lacked both guidance for setting its parameters and demonstration of its effectiveness. At its adoption, it had not even been demonstrated that there were any parameter settings that would work at all, avoiding instability or oscillations. This paper reports on extensive evaluation of IBA CCA, under different scenarios and congestion control parameters that (a) delivers the first guidance for setting CCA parameters for IBA; and (b) demonstrates that this is the first effective solution to hot spot contention published in 20 years. This result expected to be significant for standards such as IBA and IEEE 802.1/3, particularly as virtualized networks become more common.

Index Terms— congestion control, interconnection networks, InfiniBand, performance evaluation, hot spot contention

I. INTRODUCTION

Virtually all large computer installations, whether for commercial or technical computing, now consist of computing nodes connected by a multistage interconnection network. It has, however, been known since at least 1985[1] that all such networks, of any topology, are subject to catastrophic loss of throughput under traffic patterns containing a hot spot: A specific destination to which enough traffic is directed from all nodes that the destination is overloaded, backing up traffic to other nodes.

The discovery of the effects of such hot spot patterns initiated, within the parallel computing, lossless networking community, a mini-industry of solution proposals. None of them were of practical use since all explicitly or implicitly assumed *a priori* knowledge of where the hot spot was; that is, they required use of special operations at the compute nodes issuing the hot spot requests, and/or used special separate networks to handle the hot spot traffic.

These efforts ultimately died out, at least in part because hot

spots apparently did not occur in practice. In retrospect, this lack of evidence can be explained by two factors: First, particularly in the commercial processing realm, most interconnection networks have been substantially over-provisioned. It is relatively inexpensive, and avoids costly management, to simply drown the problem in excess bandwidth. Second, particularly but not exclusively in the parallel computing realm, collections of computing nodes are (perhaps temporarily) dedicated to a particular problem. This makes the traffic patterns an intrinsic part of the algorithms used. As a result, congestion and hot spots are treated as performance bugs to be analyzed and eliminated.

Unfortunately, both of those factors may be eliminated in the near future.

Over-provisioning is not a scalable solution. As the number of nodes increases, the fraction of each node's traffic required for congestion to form decreases proportionally. Now, as opposed to, say, ten years ago, the limits to Moore's law are being reached. That implies that computing capacity will be increased only by adding parallelism, and not, as today, having increasing node speed to dampen the parallelism increase needed. So parallel scaling will increase and potentially overwhelm affordable over-provisioning. Furthermore, the broad move to XML-based messaging for commercial data is estimated to increase bandwidth requirements at least 20X; this is already beginning to strain installed communication facilities.

Algorithmic predictability of network traffic is eliminated as virtualization becomes more common. When multiple virtualized networks and nodes simultaneously inhabit the same physical hardware, it is no longer the case that any one algorithm controls the traffic pattern; it becomes an overlay of several different algorithms' patterns, varying with the deployment of the virtual resources. Since the virtualization managers themselves cannot be expected to understand all systems' traffic, it becomes unpredictable.

For these reasons, networks adequate to the future demands of computing systems must incorporate effective congestion control that does not assume *a priori* knowledge by any component of the traffic patterns being imposed.

The question this paper answers, in part, is whether the congestion control architecture proposed for IBA provides this capability. We find that IBA CCA can solve the hot spot problem, which has been outstanding for 20 years, for a fair collection of traffic patterns and several network topologies, in the sense that we are able to find parameter settings that quench hot spot congestion while avoiding oscillation and other stability side-effects.

M. Gusat (phone: 41-44-724-8568, e-mail: mig@zurich.ibm.com) and W. Denzel are with IBM Research GmbH, Rüschlikon, Switzerland.

G. Pfister (e-mail: pfister@us.ibm.com) and N. Ni are with IBM Systems and Technology Group, Austin, TX USA.

D. Craddock and W. Rooney are with IBM Systems and Technology Group, Poughkeepsie, NY USA.

J. Duato is with Technical University of Valencia, Spain.

This gives IBA a unique advantage over other contenders for interconnection in the data center, such as TCP/IP over Ethernet, extensions to PCI Express, RapidIO, HyperTransport, etc.

The remainder of the paper is organized as follows: In Section II we describe congestion in lossless networks and in IB in particular. Section II briefly covers related work. Section III describes the IBA Congestion Control Architecture. Section V introduces our simulation models, networks and traffic patterns. Results, analysis and guidelines follow in Section VI. Finally, in Section VII we conclude by discussing the limitations of our current results and outline future work.

II. HOT SPOT CONTENTION

Figure 1 below illustrates the hot spot contention problem using a lossless three-stage 32 ports, 3-level fat tree using 8x8 switches, drawn in an unfolded unidirectional representation illustrating its 5 actual stages of switching. Corresponding “half” switches on the left and right hand side are collocated in the same physical switches.

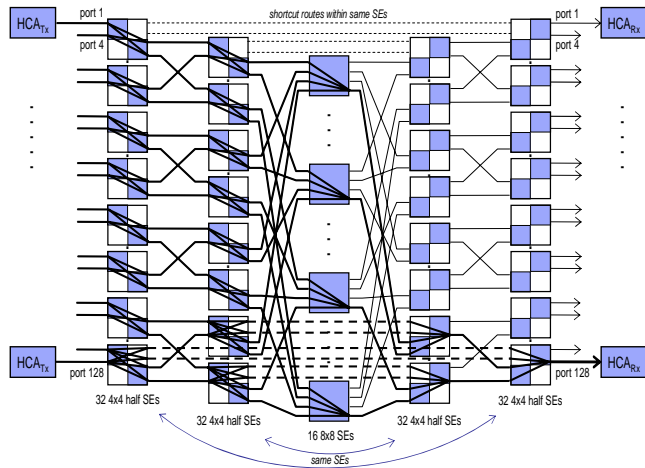


Figure 1. Illustration of Hot Spot Contention

If a sufficient fraction of all the inputs’ traffic targets one of the outputs (in the figure, the output labeled 128), that output link can become saturated. Should this happen persistently, it causes the queues in the switch feeding that link to fill up. If the traffic pattern persists indefinitely, then, no matter what techniques are used to reassign buffer space, it is all ultimately exhausted. This forces that switch’s link-level flow control to throttle back all the inputs feeding that switch (since the network is lossless). That in turn ultimately causes the next stage of switching to fill its buffer space, and so on back to the network inputs. This has been called *tree saturation* [1] or, in other contexts, *congestion spreading*.

Once the tree of saturated switches is fully formed, every packet must cross at least one saturated switch. Since the time to exit a queue grows exponentially the further a switch is from the hot destination, a majority of the delay is incurred even if only a single switch must be crossed. Hence the network as a whole suffers a catastrophic loss of throughput: Its aggregate throughput is gated by the throughput of the single hot output.

Furthermore, the amount of traffic that must target the hot output decreases proportional to network size. For example, in an 128-input and -output network like that illustrated, slightly more than 1/128 of each input’s injected traffic must target the hot output to saturate that output. However, in a 1024-input and -output, less than 0.1% of each input’s injected traffic must target the hot output for the same problem to arise.

Finally, saturation spreads very quickly; according to the analysis of [28], the tree is filled in less than 10 traversal times of the network – far too quickly for software to react in time to the problem. Naturally, the problem also dissipates slowly since all the queues involved must be emptied. Hence a hardware solution is required that reacts quickly enough to keep the tree from growing large.

From this and the lack of prior solutions it can be counted as very fortunate that the factors discussed above have for the most part mitigated this problem. It has, however, verifiably been encountered in at least one real situation involving cascaded IBM zSeries IO channel switches; it was fixed there by algorithmic modification. The authors are also aware of similar problems that appeared in other HCA networks that have not been made public.

Clearly the network topology is irrelevant to this effect; any network must contain trees like the above. The problem is a form of high-order head-of-line blocking [3, 10]. It can also be caused by flow interference [6, p. 112].

InfiniBand architecture is no more immune to this kind of contention than any other network architecture, despite already having facilities such as independently-queued virtual lanes (VLs) to keep identified flows separate from one another (facilities not present in, e.g., normal IP switching and routing). The issue addressed here, however, is not one of separating flows or fixing traffic patterns that one knows about in advance; that can always be done, using VLs or other means. The issue, rather, is avoiding the problem when it has not been predicted, but rather is caused unpredictably by unfortunate workload patterns, program bugs¹ or even attacks. It is particularly necessary to solve this in installations running multiple applications simultaneously, since its effects can seriously harm traffic that has nothing to do with the flows containing the hot traffic.

Some recently proposed techniques have been touted in the industry press as “solving” the hot spot problem, when in fact they only mitigate it if the hot spot lasts a sufficiently short time [27]. The techniques discussed here can cope with hot spots lasting for an indefinite period.

III. RELATED WORK

There is a body of research in flow and congestion control of both lossy (Ethernet, IP, ATM) and lossless networks. We concentrate here on the lossless case for reasons explained at the end of this section.

¹ Such as the recent distribution, to several hundred thousand clients of a large broadband service, of an update that inadvertently caused each client to check a central server for updates once every 2 minutes instead of once every 2 weeks.

Broadcast congestion notification combined with source quenching is used in [2]. This has limited scalability due to notification overhead, and exhibits instability introduced by the source response upon receiving an (un)quench when link delay increases.

The impact of congestion in Quadrics networks under several bit reversal patterns is shown to be significant in [10]. The authors also introduce the notion of a congestion matrix to describe the spatial distribution of congestion, a method we found particularly useful when preparing our hotspot traffic experiments.

Simplified Packet Pair (SPP) and the Alternating Static Window (ASW) flow control protocols are investigated in [9]; these are based on earlier Static Window and Packet Pair protocols. The authors show that if high throughput is desired, ASW controls average latency well; but if low throughput is acceptable, SPP can be used to maintain low latencies. Our work controls both throughput and latency under high load.

A number of IBA congestion control proposals are made in [7] but it is primarily restricted by an idealized network of only two simple switches, essentially eliminating the higher-order effects of tree saturation. Finally, the closely related [29] presents a particular subset of the work, which here will be extended to cover more networks and traffic cases.

Congestion in lossy networks, e.g., IP or ATM, is very dissimilar from hotspot phenomena [1, 2, 3, 9, 10], generally precluding the transfer of IP [13, 18-21] or ATM [8, 15] mechanisms. Key differences are:

- Most lossless network implementations have round-trip times orders of magnitude lower than lossy networks, requiring far faster response times to congestion events.
- Lossless network traffic injection is usually done with little policing or traffic shaping.
- Edge links commonly have the same rate as internal links, so there is no statistical multiplexing to smooth hot spot events.
- Critically: lossless networks, in order to maintain their lossless property, use fast, hardware-implemented link-level flow-control. This is the cause of the speed and existence of tree saturation [1].

At present there is broad industry debate about whether lossy networks in general, and TCP/IP in particular, can substitute in all cases for the lossless networks now in use. We will not enter that debate here.

IV. CONGESTION CONTROL FACILITIES IN IBA

This section overviews the mechanisms defined for IBA CCA, and then discusses some key properties.

A. IBA CCA Mechanism

Figure 2 illustrates how the IBA Congestion Control Architecture (CCA) operates. In overview, discussed in more detail below, this is a three-stage process: When congestion is detected in a switch (position 1 in the figure), the switch turns on a bit in packets called Forward Explicit Congestion Notification (FECN). When the packet arrives at the destination adapter, it responds (position 2) to the source with packets having a different bit set called Backward Explicit Congestion

Notification (BECN). When an adapter receives a BECN (position 4), it responds by throttling back its injection of packets. This reduces congestion. Over time, the source gradually reduces its throttling, which may again cause congestion to be detected. If all parameters – such as the rates of throttling and reduction in throttling over time – are appropriately set, the network should settle into a stable state with just enough congestion to keep the sources quenched. The setting of such parameters is under control of a Congestion Control Manager (CCM), which establishes their values. It may adjust them over time, but they must be set appropriately to quench congestion quickly; software intervention by the CCM is too slow to avoid the problem.

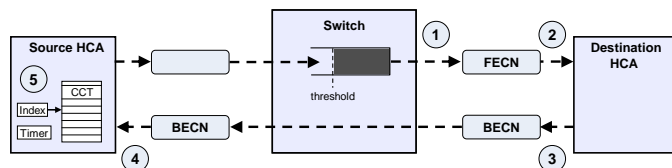


Figure 2. Technique of InfiniBand Congestion Control

In more detail:

Congestion Marking Function: InfiniBand switches detect congestion on a Virtual Lane (VL) for a given port when a relative threshold set by the CCM has been exceeded. The threshold is specified per port between 0 and 15; 0 indicates that the switch is not to mark any packets on this port, 15 specifies a very aggressive threshold. Since the switch architecture affects how the level of congestion should be determined, the exact meaning of a particular threshold setting is left to the switch manufacturer.

The switch may also be configured to only mark packets identified as the “root of congestion.” A packet is a root of congestion when the output VL has exceeded the threshold, and there are credits available. In contrast, victim (not root) congestion occurs when the output VL has exceeded the threshold and there are no credits. Packets smaller than a configurable packet size are not marked to allow vendors to avoid performance issues in switch designs and avoid marking acknowledgements (ACK) and congestion notifications (CN). Finally, the marking rate is also configurable.

When congestion is detected by the switch, it informs the destination node by setting the Forward Explicit Congestion Notification (FECN) bit of the Base Transport Header (BTH) [24] present in every IBA packet. FECN bits are passed through unchanged by all switches.

Congestion information is passed forward, and not directly back to the source, for two reasons: First, in no case do IBA switches source packets (except for management functions); it was desired to maintain this property. Second, the intent is to target the specific InfiniBand queue pair(s) in the source adapter, that is, the specific traffic flows from the adapter, that are the source of the congestion, throttling only those and not the entire adapter. Queue pair information is contained in a level of header that IBA switches do not parse or process.

Congestion Signaling: Upon receipt of a packet with the FECN bit set (position 2) the destination adapter – a “Host Channel Adapter” in IBA parlance – responds back to the

source of the packet with Backwards Explicit Congestion Notification (BECN) (3) that returns back to the specific queue pair that was the source of congestion. This may be piggy-backed on normal ACK traffic for connected communication, or may be in a special congestion notification (CN) packet for unconnected service. The HCA does necessarily process Queue Pair information, and is the source of packets in normal operation, so this is natural to its function.

Injection Rate Reduction: When the source receives a packet with the BECN bit set (4) the injection rate of the flow is reduced. The reduction can be applied either to the specific queue pair sourcing the flow, or to all QPs on the port using a particular service level (which maps to a virtual lane). The amount of reduction is controlled by a table associated with the HCA, called the Congestion Control Table (CCT), whose content is loaded by the Congestion Control Manager. Each time a BECN is received, an index into the table is incremented (position 5). The amount of the increment is the CCTI_Increase value, also set by the CCM. Each entry in the CCT defines an Inter-Packet Delay value that is inserted by the HCA between packets. As BECNs are received, the index is repeatedly incremented, indexing further into the CCT where it will usually encounter larger and larger delay values set by the manager.

Injection Rate Recovery: Each HCA contains a timer; each time it expires, a duration set by the CCM, the index into the CCT is reduced by an amount also set by the CCM. This causes use of values lower in the CCT as Inter-Packet Delays. Again, presumably the CCM has set the CCT tables so that this reduces the added inter-packet delay. Eventually, if no more BECNs are received, the index reaches zero and no delays are inserted.

B. IBA CCA Characteristics

Congestion control in IB is a closed-loop control system: congestion events affect the source injection rate, which affects the congestion. The control loop back to the injection source is required to avoid hot spots that persist over time; network buffer reallocation cannot win against persistent host injection.

However, the loop's inclusion of the destination reduces its response time significantly. This was done for reasons of cost and simplicity, since it eliminates the need for switches to inject packets or to examine higher levels of the communication stack, and reuses facilities already present in adapters which do those things.

Unfortunately, this is probably the cause of undesirable effects immediately observed in early experiments: unstable behavior, oscillations (Fig. 5b) and even throughput collapse (Fig. 5c). Particularly disturbing was the 'stiffness' of the control loop, manifested as sensitivity of the marking and source response functions to the CCA parameters.

We also suspected, and verified, that appropriate parameters vary depending on network size and topology; other parameters may also play a role. This makes it difficult for system designers and/or administrators to select a suitable set of CCA parameters. Our paper addresses this challenge. After carefully evaluating the CCA mechanisms under different

scenarios and networks, this paper delivers a guideline for the IBTA members, OEMs and for the IB community by providing recommendations to properly tune IB congestion control. Our results may be of broader use because IBA CCA scheme is representative of a wide class of networks that rely on similar types of facilities: congestion marking in the switch/router, explicit congestion signaling, and source response function to adjust the injection rate. This is E.g., TCP/IP [18-21], RapidIO [22], PCI-AS [23].

We are in particular searching for parameter settings which are sufficient, meaning that the settings are effective, fair, and stable, meaning:

- *Effective:* able to avoid, prevent, or recover from throughput collapse due to the formation of saturation trees;
- *Fair:* does no harm to innocents, fairly treating hot and cold flows as described below;
- *Stable:* dampens oscillations.

V. EXPERIMENTS WITH IB CCA: MODELS AND TRAFFIC

Our aim was to simulate networks that are as realistic as feasible.

We use stochastic system level simulations with synthetic traffic and carefully selected congestion scenarios. Actual application traces, or even execution-driven simulations, could increase accuracy. But their size and runtime limitations would preclude doing the large number of simulations needed to explore the parameter and network space. Hence we systematically pruned and explored a large simulation space with a moderate level of confidence.

In this study we considered a single VL and service level (SL), although our simulation model supports multiple VLs and SLs. This is not a significant limitation; different VLs do not interact directly in congestion in IBA because each is required to have independent buffering.

To this end, the simulation is no lower than packet-level and abstract many, potentially relevant, application-level details. We have simulated using both CSIM and OMNeT++ with different levels of accuracy of the HCA, Switch Element (SE), network topology and link-level models.

A. Simulation Models

1) Host Channel Adapter (HCA)

We model IBA's logically separate queues for each flow; each is a queue pair (QP), a send and a receive queue. When a source process is ready to send data, it posts a work queue element (WQE) to the send queue of a QP, and the HCA is notified of work to do. The QP then waits for its turn to be served by the HCA; the HCA's virtual lane arbitration is responsible for choosing the next QP to service. When a QP is picked, the data pointed to from the WQE is read from system memory into a buffer inside the HCA and sent. At the destination HCA, an incoming data packet is saved in a receive buffer and later forwarded to a system memory location specified by a receive WQE posted to the destination QP's receive queue.

2) Switch Element (SE)

Our SE model is based on a generic IB switch with shared input buffers and virtual output queuing per shared buffer.

This conceptually models commercial IB switching chips. The buffer size is chosen to be 36 maximum transmission units (MTU) per input. The output queue scheduler performs round-robin service across all the SEs. The links, between SEs or between HCAs and SEs, are intentionally short, with a round trip time normalized to $\frac{1}{4}$ MTU, in order to avoid the additional constraints of switch work conservation with long RTTs [12, 26].

3) Network Topology

Our model for multistage interconnection networks (MIN) supports multistage folded bidirectional k-ary n-fly topologies – such as Omega, Benes, *fat-tree* [16], and Banyan – built from $N \times N$ SEs as described above.

We have simulated the following IB networks, in the order listed.

- 8 nodes/ports, 3-stage, 2x2 SE: our initial trials
- 32 nodes/ports, 3-stage/2-level, 8x8 SE: chosen since 8x8 switches are more realistic; this is the network of Figure 1.
- 128 nodes/ports, 3-stage/2-level, 16x16 SE, and 128 nodes/ports, 5-stage/3-level, 8x8 SE: used to verify the results from the prior two cases, but with different configurations
- 432 nodes/ports, 5-stage/3-level, 12x12 SE, and 256 and 512 nodes/ports, with simpler SEs, our final cases.

Fig. 2 shows a 5-stage network based on 8x8 SEs, drawn in an unfolded unidirectional representation. The corresponding “half SEs” on the left and right hand side are collocated in the same physical SEs. Hence connections to nearby ports on the same SE can take shortcut routes (dashed in Fig. 2) within the same SE and do not need to traverse the middle stage(s). Similar topologies are used in commercial InfiniBand switch systems.

Here we report only a few representative simulations.

B. Traffic

The traffic injected by each source HCA has two components: uniformly distributed background traffic, also referred to as cold traffic; and hot traffic. During a defined congestion period, a predetermined fraction of the HCAs all divert a fixed portion of their traffic to a defined hot port (in the following, port 31). Traffic is injected into the system as packets of fixed MTUs equal to 2KB, except for the ACK and ECN packets that are used to measure the impact of shorter packets. In the absence of congestion the inter-packet time is negative-exponentially distributed.

This pattern is representative of a large class of commercial and HPC applications that use collective and synchronization operations, or which frequently exercise reliable multicast transmissions.

To describe the traffic patterns we use the following terms:

- *Hotspot Degree (HSD)* is the number of sources that contribute to the hotspot.
- *Hotspot Severity (HSV)* is the total amount of resource oversubscription at the hot destination, caused by both the hot and the background traffic.

- *Offered Load:* The total load, hot spot plus background traffic, sent to the network.

For each IB network topology of size N we show the results of the four cases listed in Table 1. All have HSV of 300%, with varying HSD and total offered load. based on the combinations of medium and high background loads (0.5 and 0.9) and small and large HSD for a given HSV of 300%: (We have simulated other cases which confirm the analysis given here, to be presented in a forthcoming more comprehensive paper).

Table 1

TRAFFIC CASE	1.	2.	3.	4.
Offered load	0.5	0.5	0.9	0.9
Hotspot degree (HSD)	3	N (all sources)	3	N (all sources)

In addition, for each of the considered IB networks topologies and each of the four traffic cases we perform a sensitivity analysis by varying certain parameters from a starting point as follows:

Table 2

Parameter	Starting Point
a. Switch output queue threshold (<i>sw_th</i>)	90% of the SE input buffer size (shared input buffer)
b. Maximum IPD (<i>max_ipd</i>)	Worst case round-robin fairness: the IPD must be large enough to allow <u>all the other inputs</u> to inject one MTU each.
c. IPD table index increment (<i>ipd_idx_incr</i>)	1 ... 128.
d. IPD recovery timer (<i>rec_time</i>)	Derived from the measured BECN inter-arrival times at source nodes.

VI. SIMULATION RESULTS

A. Input-generated hotspots in a 32-port 3-stage fat-tree network

Fig. 3 shows the simulation results obtained for the four cases considered in Table 1 for a network size of 32. The first row shows the throughput of the first 32 output ports (i.e. all nodes) normalized over the simulation time, *without* congestion control. The second row shows the same *with* congestion control. The third row shows the resulting IPDs.

The following congestion control parameters were used, as derived from a comprehensive parameter sensitivity analysis: switch output queue threshold $sw_th = 90\%$ of switch input buffer size; maximum IPD table entry $max_ipd = 21\mu s$; IPD table index increment $ipd_idx_incr = 5$; IPD recovery time $rec_time = 10.5\mu s$.

Without congestion control, the top row, the throughput of the hotspot port (#31) saturates during the congestion period, while the throughput for all other output ports drops. Obviously a congestion tree has built up that hurts all the flows, hot or cold. After the hotspot period the aggregate throughput recovers slowly because all the queues in the system are still backlogged and must be emptied to the prior level.

With congestion control, the middle row, the throughput collapse problem is effectively solved. The slightly reduced throughput observed during the congestion period is correctly attributed to the portion of the offered load that is being re-directed to the hotspot.

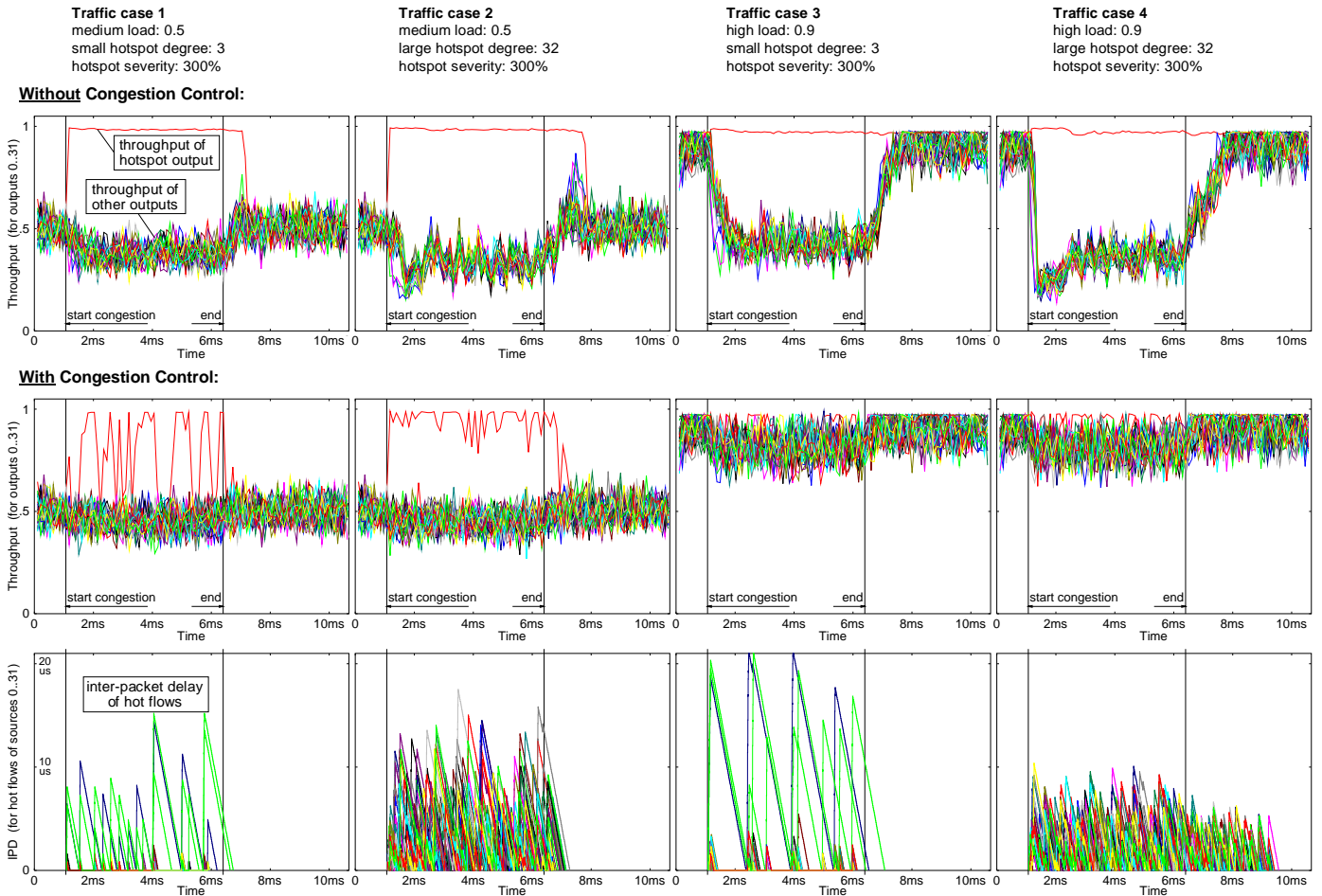


Figure 3. Simulation results for 4 cases of input-generated hotspots in a 32-port 3-stage fat-tree network

The IPD charts, the bottom row, show that under congestion control only the hot flows reach high IPD values.

In order to find the good parameter settings, we systematically varied the relevant parameters for each case during a few hundreds of simulation runs. Fig. 4 shows the throughput results for the most critical case of high load and large HSD when the IPD index step is too low or high; or and the IPD recovery timer is too low or high.

If the IPD index step is too low *or* too high there are oscillations (4a, 4b). This means that one cannot, for example,

simply set the index step to a high (or low) value and be sure that hot spots are all appropriately quenched.

If the IPD recovery timer is too low, there is throughput collapse (4c). If it is too high, the hot flows show an unnecessary loss of throughput (4d).

B. Input-generated hotspots in a larger 128-port 5-stage fat-tree network

Fig. 5 shows simulation results from a 128-port 5-stage fat-tree network, with the same data display as used above, except

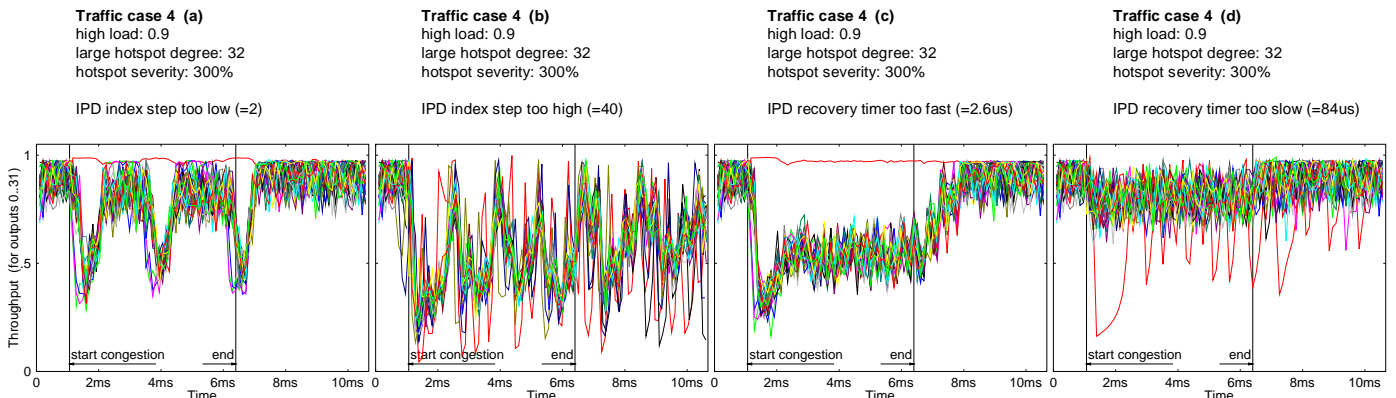


Figure 4. Simulation result examples for out-of-range parameters

