Understanding Congestion in High Performance Interconnection Networks Using Sampling

Philip Taffet, John Mellor-Crummey

Abstract

Goal: Enable tools that help developers analyze and tune the communication performance of their applications

Use a novel sampling method to capture information about where, when, and why congestion is occurring.

This sampling-based technique collects information about the path a packet takes and congestion it encounters. This strategy can distinguish problems with:

- an application's communication patterns,
- its mapping onto a parallel system,
- and outside interference.

A variant of this scheme requires only 5-6 bits of information in a monitored packet, making it practical for next-generation networks.

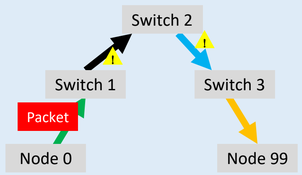
Reservoir Sampling

Hop count ← *Hop count + 1*
With probability $1/Hop\ count$:
Link ← link ID of outgoing link
Congested? ← is outgoing link congested

1

Packet

Source: 0, Destination: 99, Packet ID, Other headers
Traffic Sample Link: Congested?: **NO** Hop count: **1**
Packet Data

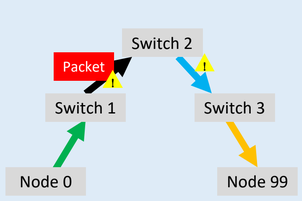


When creating the packet:
Node 0 initializes:
Hop count ← 1
Link ← *Green*
Congested? ← No

2

Packet

Source: 0, Destination: 99, Packet ID, Other headers
Traffic Sample Link: Congested?: **NO** Hop count: **2**
Packet Data



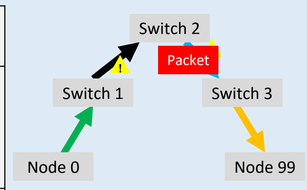
As the packet leaves Switch 1:
Switch 1 updates:
Hop count ← *2*
With probability $1/2$:
Link ← Black
Congested? ← Yes

In this example, the random choice was not to update at this hop.

3

Packet

Source: 0, Destination: 99, Packet ID, Other headers
Traffic Sample Link: Congested?: **YES** Hop count: **3**
Packet Data

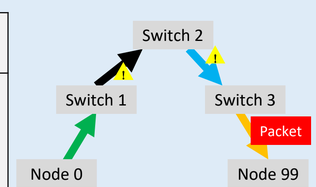


As the packet leaves Switch 2:
Switch 2 updates:
Hop count ← *3*
With probability $1/3$:
Link ← Blue
Congested? ← Yes

4

Packet

Source: 0, Destination: 99, Packet ID, Other headers
Traffic Sample Link: Congested?: **YES** Hop count: **4**
Packet Data



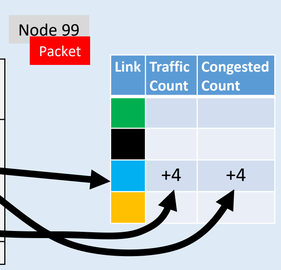
As the packet leaves Switch 3:
Switch 3 updates:
Hop count ← *4*
With probability $1/4$:
Link ← Gold
Congested? ← No

In this example, the random choice was not to update at this hop.

5

Packet

Source: 0, Destination: 99, Packet ID, Other headers
Traffic Sample Link: Congested?: **YES** Hop count: **4**
Packet Data



Link	Traffic Count	Congested Count
	+4	+4

When the packet arrives:
Node 99 updates:
Δ = Hop count
TrafficCount[Link] += Δ
If Congested?
CongestedCount[Link] += Δ

If N packets traverse this path, approximately $\frac{N}{4}$ of them have each link ID when they arrive at Node 99. Thus the NIC's count for each link ID will be approximately N, which reflects the fact that N packets passed through the link.

We aggregate these counts from all nodes and compute for each link

$$\text{congested fraction} = \text{congested count}/\text{traffic count}$$

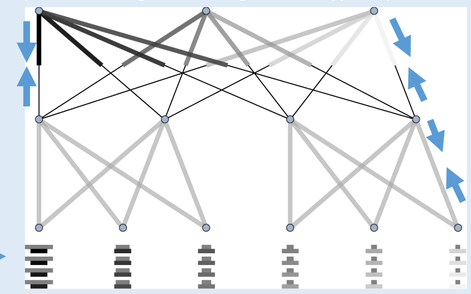
Congested Fraction Plots

By plotting the congested fraction for each link on, we can understand network congestion visually. We can interpret these diagrams to diagnose the type of problem.

Thicker = More traffic
Darker = More congested

Top half of edge corresponds to down-pointing link

Links to compute nodes shown as boxes:



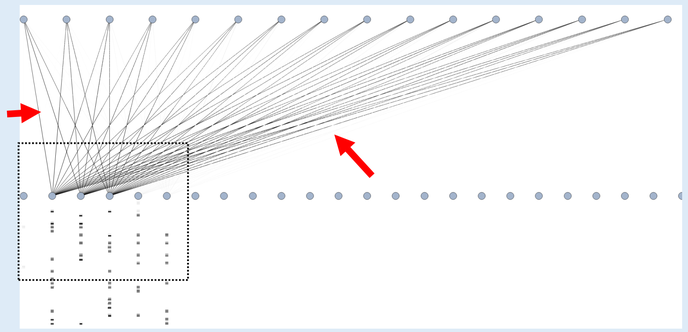
Diagnosing the Problem

Congestion often backs up, forming tree-like patterns. We look for the root of the tree.

- Congestion rooted in interior of the network
  - May be a problem with the mapping onto the physical network topology
  - Try optimizing the mapping
- Congestion rooted at an endpoint
  - Communication pattern problem
  - Code changes likely needed
- Dark and thin links in the congested fraction plot
  - External interference from background traffic

Simulation Case Study with pF3D

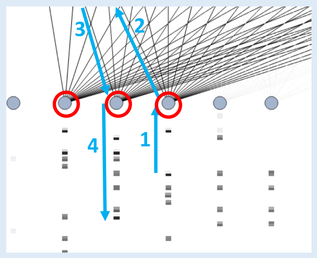
- pF3D is a laser-plasma interaction multi-physics code from LLNL.
- Under heavy background congestion, pF3D's parallel FFTs run 24% slower.
- The section takes 24.6s with no congestion and 30.4s with heavy background congestion.
- Can we understand why? Can we improve the performance?



- Links from leaf switches to 2nd level switches are dark and thin
- High congested fraction but low traffic

- pF3D drives an average of 2.0 Gbps per link
- Congestion must be due to background traffic

If we assume the external interference will remain constant, we can also look for a second diagnosis.



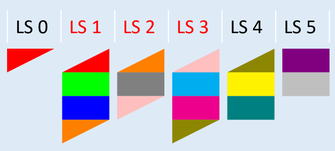
Link	Congested fraction	
1	0.86	
2	1.00	
3	0.94	← Root
4	0.06	

○ Root of a congestion tree

- Root is in network interior
- Try to solve congestion with mapping
- In particular, we need a mapping that reduces traffic entering circled switches

- On this problem size, much of pF3D's communication occurs in 4-node groups
- By shifting the node numbering, we prevent groups from communicating over congested links.

We compare the mappings by representing each group of 4 nodes by a different colored rectangle



Original mapping

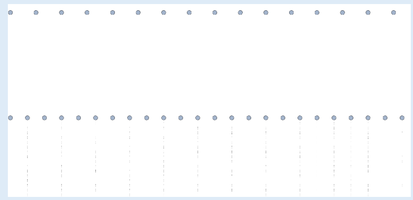
Shifted mapping

	Time (s)	% difference
Congestion, default mapping	30.40	-
Congestion, shifted mapping	25.50	16.1%
No Congestion	24.60	19.1%

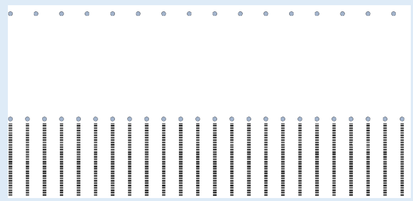
Application-centric

Storing performance information in packets lets us correlate it back to the application.

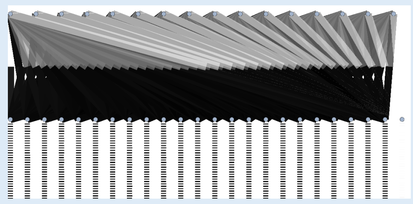
For example, unless miniGhost uses a mapping that provides good locality in each grid dimension, its congestion varies dramatically between different communication phases. We split data based on MPI tag because phases overlap in time.



$+x$ phase
Good locality
Very little inter-node traffic
Very little congestion



$+y$ phase
Okay locality
Heavy congestion, but contained to links to and from compute nodes



$+z$ phase
Poor locality
Links to 2nd level severely congested
Requires more bisection bandwidth than tapered network provides

Probabilistic Encoding of Network Traffic

High Level Idea

Let H be a hash function that returns 1 bit
Replace main algorithm and packet with:

Hop count ← *Hop count + 1*
With probability $1/Hop\ count$:
Link ← H(packet ID, link ID of outgoing link)
Congested? ← is outgoing link congested

Packet

Source: 0, Destination: 99, Packet ID, Other headers	
Traffic Sample	
Link hash: **H(packet ID,)**	1 bit
Congested?: **NO**	1 bit
Hop count: **1**	3-4 bits
Packet Data	

Replace reconstruction step with :

5'

Packet

Source: 0, Destination: 99, Packet ID, Other headers
Traffic Sample Link hash: **0=H(packet ID,)** Congested?: **YES** Hop count: **4**
Packet Data

Link	H(ID, _)	Traffic Count	Congested Count
	1	-4	-4
	0	+4	+4
	0	+4	+4
	1	-4	-4

When the packet arrives:
Node 99 updates for each *link* that packet could have gone through:
If H(packet ID, link) == link hash
Δ = hop count
else
Δ = -1 hop count*
TrafficCount[link] += Δ
If Congested?
CongestedCount[link] += Δ

Analysis

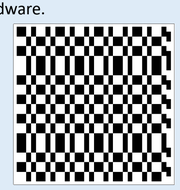
Fix a link ID d. Packet i contains H(packet ID$_i$, d_i). For analysis, split the packets into three groups:
Packets where $d = d_i$, so H(packet ID$_i$, d) = H(packet ID$_i$, d_i).
Packet **would** have contained d if non-probabilistic
+1 for each of these
Packets where $d \neq d_i$, but H(packet ID$_i$, d) = H(packet ID$_i$, d_i).
Packet **would not** have contained d if non-probabilistic
+1 for each of these
Packets where $d \neq d_i$, and H(packet ID$_i$, d) ≠ H(packet ID$_i$, d_i).
Packet **would not** have contained d if non-probabilistic
-1 for each of these

We can't distinguish group I and II, but group II cancels out group III
Expected value of count is the number of packets that would have contained d

Hash Function

We use a hash function based on multiplication in the field $\mathrm{GF}(2^k)$. This function satisfies the required properties and is easy to compute in hardware.

H(14, 1..1024)



H(15, 1..1024)

- H is not random, but much easier to compute than a cryptographic hash
- H is balanced between 1 and 0
  - False positives and true negatives cancel out
- $H(id_1, d)$ and $H(id_2, d)$ are uncorrelated
  - Counts for one link ID will not bleed into other link IDs.