I have two [multi-stage] slides [after this one]

Two problems that I spend a lot of time thinking about

• Both have to do with *bounded buffering*

Trends

• Software design is a programming task. Hardware design is becoming more of a programming task.

• Hardware is concurrent and parallel. Software is becoming more concurrent and parallel.

• As the two disciplines grow towards each other, I speculate that *bounded buffering* is the fundamental division between them that will remain after the dust settles.
A Transactor Network

count up

>...3,2,1,0>

count up

>...3,2,1,0>
A Transactor Network

This addition causes no problems

\[ \ldots 3, 2, 1, 0 \to 16, 10, 5, 1 \to \ldots \]
Another Transactor Network

>$\ldots9,8,3,1> \rightarrow \text{sorted merge} \rightarrow >\ldots9,8,7,3,2,2,1,0>

>$\ldots7,2,2,0$
Another Transactor Network

This addition causes BIG PROBLEMS

\[ \ldots 9,8,3,1 > \]
\[ \ldots 7,2,2,0 > \]
\[ + \]
\[ \ldots 16,10,5,1 > \]
\[ \ldots 9,8,7,3,2,2,1,0 > \]
No finite amount of buffering on the input channels is sufficient. \( \exists \) Eloise can always beat \( \forall \) Belard.
Clogging Networks

• A network at *transactor granularity* cannot afford:
  • Retransmit buffers at every sender
  • Reordering buffers at every receiver
  • Sequence numbers and timers
  • Bufferlock recovery (RAW/Tilera/Parks)

• Therefore, the network cannot simply imitate the Internet
  • Cannot drop packets in response to congestion
  • Must offer reliable delivery (and wrap unreliable links)
  • *But this raises serious clogging issues!*
Clogging Networks

• Suppose \texttt{Adder} waits for pairs of items to become available and consumes them pairwise.
  
  • If items sent from the two outputs are interleaved properly, this will work.
Clogging Networks

- Suppose \texttt{Adder} waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.
Clogging Networks

- Suppose \texttt{Adder} waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.
Clogging Networks

• Suppose \texttt{Adder} waits for pairs of items to become available and consumes them pairwise.
  
  • If items sent from the two outputs are interleaved properly, this will work.
Clogging Networks

- Suppose Adder waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.
Clogging Networks

• Suppose **Adder** waits for pairs of items to become available and consumes them pairwise.
  
  • If items sent from the two outputs are interleaved properly, this will work.
Clogging Networks

- Suppose \texttt{Adder} waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.
Clogging Networks

• Suppose \texttt{Adder} waits for pairs of items to become available and consumes them pairwise.
  
  • If items sent from the two outputs are interleaved properly, this will work.
Clogging Networks

- Suppose Adder waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.
Clogging Networks

• Suppose Adder waits for pairs of items to become available and consumes them pairwise.
  • If items sent from the two outputs are interleaved properly, this will work.

![Diagram of clogging networks with FIFO and Adder components]
Clogging Networks

• Suppose Adder waits for pairs of items to become available and consumes them pairwise.
  • If items sent from the two outputs are interleaved properly, this will work.
Clogging Networks

• Suppose \texttt{Adder} waits for pairs of items to become available and consumes them pairwise.
  • If items sent from the two outputs are interleaved properly, this will work.
Clogging Networks

- Suppose Adder waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.
Clogging Networks

- Suppose \textit{Adder} waits for pairs of items to become available and consumes them pairwise.
  - If items sent from the two outputs are interleaved properly, this will work.
Clogging Networks

• Suppose Adder waits for pairs of items to become available and consumes them pairwise.

  • If items sent from the two outputs are interleaved properly, this will work.
Clogging Networks

• Suppose Adder waits for pairs of items to become available and consumes them pairwise.
  • If items sent from the two outputs are interleaved properly, this will work.
Clogging the Network

• Delays through the network are unpredictable
  • What if all packets from one source arrive before any from the other?
  • Packets “back up into” the shared portion of the network
    • Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).
Clogging the Network

- Delays through the network are unpredictable
- What if all packets from one source arrive before any from the other?
- Packets “back up into” the shared portion of the network
- Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).

![Network Diagram]

Wednesday, September 9, 2009
Clogging the Network

- Delays through the network are unpredictable
  - What if all packets from one source arrive before any from the other?
  - Packets “back up into” the shared portion of the network
    - Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).

Fifo

out

network

Fifo

out

Adder

in1

in2
Clogging the Network

• Delays through the network are unpredictable
  • What if all packets from one source arrive before any from the other?
  • Packets “back up into” the shared portion of the network
    • Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).
Clogging the Network

• Delays through the network are unpredictable
  • What if all packets from one source arrive before any from the other?
  • Packets “back up into” the shared portion of the network
    • Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).
Clogging the Network

- Delays through the network are unpredictable
  - What if all packets from one source arrive before any from the other?
  - Packets “back up into” the shared portion of the network

- Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).

\[\text{Fifo} \rightarrow \text{network} \rightarrow \text{Adder}\]
Clogging the Network

• Delays through the network are unpredictable
  • What if all packets from one source arrive before any from the other?
  • Packets “back up into” the shared portion of the network
    • Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).
Clogging the Network

• Delays through the network are unpredictable
  • What if all packets from one source arrive before any from the other?
  • Packets “back up into” the shared portion of the network
    • Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).
Clogging the Network

• Delays through the network are unpredictable
  • What if all packets from one source arrive before any from the other?
  • Packets “back up into” the shared portion of the network
    • Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).

Weekend, September 9, 2009
Clogging the Network

- Delays through the network are unpredictable
- What if all packets from one source arrive before any from the other?
- Packets “back up into” the shared portion of the network
- Usually leads to deadlock. Dally calls this “high level deadlock,” throws his hands up in the air (sec 14.1.5).
Flow Control

- In order to solve the clogging problem, we must establish *flow control*.
- Ensure that the number of items in flight towards a given destination never exceeds the amount of buffering dedicated to that destination.
Flow Control

• In order to solve the clogging problem, we must establish *flow control*.

  • Ensure that the number of items in flight towards a given destination never exceeds the amount of buffering dedicated to that destination.
Flow Control

• In order to solve the clogging problem, we must establish *flow control*.

• Ensure that the number of items in flight towards a given destination never exceeds the amount of buffering dedicated to that destination.
Flow Control

• In order to solve the clogging problem, we must establish *flow control*.

  • Ensure that the number of items in flight towards a given destination never exceeds the amount of buffering dedicated to that destination.
Flow Control

• In order to solve the clogging problem, we must establish *flow control*.
  
  • Ensure that the number of items in flight towards a given destination never exceeds the amount of buffering dedicated to that destination.
Flow Control

• In order to solve the clogging problem, we must establish *flow control*.

  • Ensure that the number of items in flight towards a given destination never exceeds the amount of buffering dedicated to that destination.

*At most one packet in the network at any time*