Deadlock-free Packet Switching

CS60002: Distributed Systems

Pallab Dasgupta
Dept. of Computer Sc. & Engg.,
Indian Institute of Technology Kharagpur
Store and forward deadlock

Buffer-size = 5

Node $s$ sending 5 packets to $v$ through $t$
Node $v$ sending 5 packets to $s$ through $u$
Model

- The network is a graph $G = (V, E)$
- Each node has $B$ buffers

Moves:
- **Generation.** A node $u$ creates a new packet $p$ and places it in an empty buffer in $u$. Node $u$ is the source of $p$.
- **Forwarding.** A packet $p$ is forwarded from a node $u$ to an empty buffer in the next node $w$ on its route.
- **Consumption.** A packet $p$ occupying a buffer in its destination node is removed from the buffer.
The packet switching controller has the following requirements:

1. The consumption of a packet (at its destination) is always allowed.
2. The generation of a packet in a node where all buffers are empty is always allowed.
3. The controller uses only local information, that is, whether a packet can be accepted in a node $u$ depends only on information known to $u$ or contained in the packet.
Solutions

• **Structured solutions**
  – **Buffer-graph based schemes**
    • The destination scheme
    • The hops-so-far scheme
    • Acyclic orientation based scheme

• **Unstructured solutions**
  – Forward count and backward count schemes
  – Forward state and backward state schemes
A buffer graph (for, $G, B$) is a directed graph $BG$ on the buffers of the network, such that

1. $BG$ is acyclic (contains no directed cycle);
2. $bc$ is an edge of $BG$ if $b$ and $c$ are buffers in the same node, or buffers in two nodes connected by a channel in $G$; and
3. for each path $\pi \in P$ there exists a path in $BG$ whose image is $\pi$.
   - $P$ is the collection of all paths followed by the packets – this collection is determined by the routing algorithm.
Suitable buffer and guaranteed path

Let $p$ be a packet in node $u$ with destination $v$.

- A buffer $b$ in $u$ is suitable for $p$ if there is a path in $BG$ from $b$ to a buffer $c$ in $v$, whose image is a path that $p$ can follow in $G$.

- One such path in $BG$ will be designated as the guaranteed path and $nb(p, b)$ denotes the next buffer on the guaranteed path.

- For each newly generated packet $p$ in $u$ there exists a designated suitable buffer, $fb(p)$ in $u$. 
The buffer-graph controller

1. The generation of a packet $p$ in $u$ is allowed iff the buffer $fb(p)$ is free. If the packet is generated it is placed in this buffer.

2. The forwarding of a packet $p$ from a buffer in $u$ to a buffer in $w$ is allowed iff $nb(p, b)$ (in $w$) is free. If the forwarding takes place $p$ is placed in $nb(p, b)$.

The buffer-graph controller is a deadlock-free controller.
The Destination Scheme

- Uses $N$ buffers in each node $u$, with a buffer $b_u[v]$ for each possible destination $v$
  - It is assumed that the routing algorithm forwards all packets with destination $v$ via a directed tree $T_v$ rooted towards $v$.

The buffer graph is defined by $BG = (B, E)$, where $b_u[v_1]b_w[v_2] \in E$ iff $v_1 = v_2$ and $uw$ is an edge of $T_{v_1}$.

There exists a deadlock-free controller for arbitrary connected networks that uses $N$ buffers in each node and allows packets to be routed via arbitrarily chosen sink trees
The Hops-so-far Scheme

- Node $u$ contains $k + 1$ buffers $b_u[0], \ldots, b_u[k]$.
- It is assumed that each packet contains a hop-count indicating how many hops the packet has made from its source.

The buffer graph is defined by $BG = (B, E)$, where $b_u[i]b_w[j] \in E$ iff $i + 1 = j$ and $uw$ is an edge of the network.

There exists a deadlock-free controller for arbitrary connected networks that uses $D+1$ buffers in each node (where $D$ is the diameter of the network), and requires packets to be sent via minimum-hop paths.
Acyclic Orientation based Scheme

**Goal:** To use only a few buffers per node

- An acyclic orientation of $G$ is a directed acyclic graph obtained by directing all edges of $G$.
- A sequence $G_1, \ldots, G_B$ of acyclic orientations of $G$ is an *acyclic orientation cover of size B* for the collection $P$ of paths if each path $\pi \in P$ can be written as a concatenation of $B$ paths $\pi_1, \ldots, \pi_B$, where $\pi_i$ is a path in $G_i$.

- A packet is always generated in node $u$ in buffer $b_u[1]$.
- A packet in buffer $b_u[i]$ that must be forwarded to node $w$ is placed in buffer $b_w[i]$ if the edge between $u$ and $w$ is directed towards $w$ in $G_i$, and to $b_w[i + 1]$ if the edge is directed towards $u$ in $G_i$.

*If an acyclic orientation cover for $P$ of size $B$ exists, then there exists a deadlock-free controller using only $B$ buffers in each node.*
Forward and Backward-count Controllers

Forward-count Controller:

- For a packet $p$, let $s_p$ be the number of hops it still has to make to its destination ($0 \leq s_p \leq k$)
- For a node $u$, $f_u$ denotes the number of free buffers in $u$ ($0 \leq f_u \leq B$)

The controller accepts a packet $p$ in node $u$ iff $s_p < f_u$.

*If $B > k$ then the above controller is a deadlock-free controller*

Backward-count Controller:

- For a packet $p$, let $t_p$ be the number of hops it has made from its source

The controller accepts a packet $p$ in node $u$ iff $t_p > k - f_u$. 
Forward and Backward-state Controllers

Forward-state Controller:
- For a node $u$ define (as a function of the state of $u$) the state vector as $(j_0, \ldots, j_k)$, where $j_s$ is the number of packets $p$ in $u$ with $s_p = s$.

The controller accepts a packet $p$ in node $u$ with state $(j_0, \ldots, j_k)$ iff:

$$\forall i, 0 \leq i \leq s_p : i < B - \sum_{s=i}^{k} j_s$$

*If $B > k$ then the above controller is a deadlock-free controller*

Backward-state Controller:
- Define the state vector as $(i_0, \ldots, i_k)$, where $i_t$ is the number of packets in node $u$ that have made $t$ hops.

The controller accepts a packet $p$ in node $u$ with state $(i_0, \ldots, i_k)$ iff:

$$\forall j, t_p \leq j \leq k : j > \sum_{t=0}^{j} i_t - B + k$$
Forward-state versus Forward-count

- Forward-state controller is more liberal than the forward-count controller

- Every move allowed by the forward-count controller is also allowed by the forward-state controller