IMPLEMENTATION AND EVALUATION OF SHUFFLENET, GEMNET AND DE BRUIJN GRAPH LOGICAL NETWORK TOPOLOGIES

Tekiner Firat, Ghassemlooy Zabih, Al-khayatt Samir, Thompson Mark
Optical Communications Research Group, School of Engineering, Sheffield Hallam University, Pond Street, Sheffield, S1 1WB. U.K. Email: ftekiner@ieee.org

Abstract – In this paper, a generic model for implementing logical interconnection topologies in software has been proposed in order to investigate the performance of the logical topologies and their routing algorithms for packet based synchronous networks. This model is generic for synchronous transfer modes and therefore can be used in implementing any logical topology by using any programming language. Three topologies have been investigated and implemented namely: Shufflenet, De Bruijn graph and Gemnet. Results for the average packet delay show that the De Bruijn graph performed the worst. Also, it is observed that the De Bruijn graph makes use of buffering more efficiently compared to the other algorithms.

Key Words- Shufflenet, Gemnet, De Bruijn Graph, Interconnection Networks, Routing, Multi-hop Logical Topolgy, Network Topology.

1. Introduction

There are four major issues when selecting and designing a logical network: switching method, network operation mode, control strategy and network topology with its routing algorithm [1]. As for switching, there are two main techniques, circuit switching and packet switching [2, 3]. The former switches in the order of seconds, whereas the latter switches in the order of microseconds. Packet switching is widely used in today's networks, which provides high bandwidth utilisation by means of dynamically allocating bandwidth on demand [3]. Operation mode can be either asynchronous or synchronous, in synchronous mode; packets are sent when a time slot is available, whereas in asynchronous networks; packets are sent when a link is available. A typical network consists of several nodes (switching elements) and interconnection links which could be controlled either by distributed nodes or by a centralised node(s). In distributed control on receiving a packet, routing decision are made at each node independently, whereas in centralised control, routing decision is made by a central processor(s) and is based on the current situation of the network. Network topology is the way that the nodes are connected to each other. Although the single-hop network is the most basic, it requires a pre-transmission phase and fast tuning transceivers (as in WDM [4]). A multihop [5] network overcomes these drawbacks and it can be regular or arbitrary [6]. A network topology is regular if node connections are based on a specific connectivity pattern which is also called an interconnection network. However, in arbitrary network topologies, there is no fixed predefined pattern as it name suggests. The main advantage of the arbitrary network is that they can be dynamic and scalable, therefore, new nodes can be added or removed without making any changes in the network structure as in the Internet. On the other hand, regular network topologies, with well defined connectivity patterns is the alternative option for more static environments (such as router and multiprocessor architectures, etc…) as they provide simpler node structures and routing schemes. Therefore, interconnection networks can be either used as an underlying physical topology itself as in multiprocessor parallel networks [7] or as a logical topology based on an underlying physical topology. The idea of imposing a logical topology for broadcast networks (i.e. star, tree bus) was first imposed by [8] to maintain and improve the performance of the network. There are a number of logical network topologies that have been applied for multihop networks [9, 10, 11]. In multihop packet-routing networks, packets traverse a number of intermediate nodes until they arrive at their destination [9]. The routing algorithm used at each node is identical; a decision made is the best possible port allocation to the incoming packets. This decision is made independently, without regard for the routing decisions made at the rest of the nodes within the network. However, each network topology has its own specific routing algorithm. Therefore, these algorithms are much simpler than the routing algorithms used in arbitrary networks, where in most cases routing tables are used [2].

A Network's topology and its routing algorithm are the key factors in determining networks performance. Therefore, the focus here is on finding the suitable network topology with its routing algorithm for a packet based synchronous network. A generic model was designed in order to be able to implement synchronous logical network topologies with their routing algorithms in software. In this paper, we will show the average number of hops needed for a packet to travel from source to destination with respect to the system load and network size.
2. Logical Network Topologies

Logical Network topology describes the way that packets travel within the network independently from the physical interconnection of the network devices [8]. By using suitable topologies, network performance can be improved and resources can be used efficiently. Three topologies together with their routing algorithms namely: Shufflenet [12], De Bruijn Graph [13] and Gemnet [14, 15] have been considered as follows:

2.1 Shufflenet

Shufflenet is classified as a regular multihop network topology which is a symmetric and homogeneous, i.e., the network looks the same from any node [9]. Thus, the number of hops from a source to destination is minimised. Due to its cylindrical connectivity pattern (see Figure 1) routing is reliable and fault tolerant. Moreover, deflection routing can be applied straightforward with promising results.

A \((p,k)\) Shufflenet consists of \(N = kp^k\) \(k = 1,2,3,...\) nodes arranged in \(k\) columns of \(p\) nodes each and with the last \((k\)-th\) and first columns connected together cylindrically where \(p\) is in and out degree of the nodes. Figure 1 shows a 24 nodes Shufflenet configuration. In general, node \((r,c)\) is connected to nodes \((r \pm p^n, c \pm p^n)\) \((r,c) \in \mathbb{Z}^2\), \(r \neq 0\) and \(c \neq 0\).

In simple static Shufflenet routing, there is only one path from source to destination, which is fixed [16, 17]. However, if a packet cannot reach its destination in one pass (i.e. it is \(k\)-hops away) then it will take one of the many minimum hop routing paths available. A simple adaptive algorithm exploits this multiplicity of potential routing paths, by routing packets via less-congested paths whenever possible [17]. Generally there are two kinds of packets defined in adaptive routing: TYPE-S packets that have a single minimum hop path from its current location to destination and TYPE-M packets that have multiple minimum hop paths from its current location to destination [17]. TYPE-M packets have no preference on which output port it goes to. Pseudo code for the routing algorithm is given in Figure 2.

\[
\text{Find distance}(X) \text{ between source nodes } (r^s,c^s) \text{ and destination node } (r^d,c^d) \\
X = \begin{cases} 
(k + c^d + e^s) \mod k, & e^s \neq c^s \\
k, & e^d = c^s 
\end{cases} \\
\text{where } r \text{ and } c \text{ is the column and row coordinates} \\
(r = 0,1,2,...p^{k-1} \text{ and } c = 0,1,2,...k-1) \\
\text{Define Packet Type } (S \text{ or } M) \\
\text{If } r^s \neq r^d \text{ then Packet Type is } S \\
\text{Else Packet Type is } M \\
\text{If Packet Type is } M \\
\text{Then forward packet to any of the available port} \\
\text{ElseRecode} = r^s \mod k, (r^s \pm p, (c^s \pm 1) \mod k) \\
\text{Figure 2: Pseudo code for Shufflenet routing algorithm}
\]

2.2 De Bruijn Graph

De Bruijn Graphs can support much larger numbers of nodes than the same degree Shufflenets by having the same average number of hops [13]. Moreover, it retains the simple addressing and routing properties of Shufflenets.

A \(G(\Delta,D)\) is a directed De Bruijn graph which consist of \(N = \Delta^D\) nodes with the set of \(0,1,2,...\Delta-1\) nodes where there is an edge from node \((a_1,a_2,...,a_D)\) to node \((b_1,b_2,...,b_D)\) if and only if
\[ b_i = a_i + 1 \]
\[ a_0, b_i \in \{0, 1, 2, \ldots, \Delta-1\} \quad 1 < i-D-1 \]

\( D \) is the diameter of the De Bruijn graph with a deflection penalty \( D \). Each node has in and out degree \( \Delta \), and \( \Delta \) nodes (i.e., 000, 111) have self-loops (self-loops exists in the logical graph but does not exist in the physical network configuration). The De Bruijn graph structure is inherently asymmetric due to the nodes with self-loops (see Figure 3) [13].

There is one-to-one correspondence between the connectivity of the nodes in the De Bruijn graph \( G (\Delta, D) \) with all the possible states of a \( \Delta \) shift register of length \( D \). If state \( b \) can be reached from state \( a \) in one shift operation in the shift register then there is an edge from node \( a \) to \( b \). Therefore, the De Bruijn graph can be seen as the state transition diagram of the shift register [18]. A node in the De Bruijn graph can be represented by a sequence of \( D \) digits as defined in the shift register analogy [18]. An edge from node \( A \) to node \( B \) can be represented by a string of \( D + 1 \) digit. Consequently, any path in the graph of length \( k \) from source to destination nodes can be represented by a string \( D + k \) digits. The first \( D \) digits represent the source node and the last \( D \) digits represent the destination node. To find the routing code from source to destination, the algorithm determines the longest suffix \(^1\) of source which appears as a prefix of destination. In order to find this, two operations, shift-match (\( i, A, B \)) and merge (\( i, A, B \)), based on the shift register analogy are defined for De Bruijn graphs as follows:

Given two strings \( A = (a_1, a_2, \ldots, a_D) \) and \( B = (b_1, b_2, \ldots, b_D) \), shift-match (\( i, A, B \)) operation is true if and only if \((a_{i+1}, a_{i+2}, \ldots, a_{i+D})\) to node \( B = (b_1, b_2, \ldots, b_D)\)

Find \( i \) between nodes \( A \) and \( B \) by using shift-match
\( i, A, B \) operation
\( i = 0; \)
while (shift-match (\( i, A, B \)) is False)
\( i = i++; \)
end while
Routing code (shortest path) between nodes \( A \) and \( B \) is given by merge operation (\( i, A, B \))
Routing code = merge (\( i, A, B \))

Figure 4: Pseudo code for De Bruijn graph routing algorithm.

A string (or a sequence) of length \( D + i \) given by \((a_1, \ldots, a_D, b_{D+1}, \ldots, b_D)\) is defined as merge (\( i, A, B \)), where \( 0 \leq i \leq D \). Note that \( i \) also gives the hop distance (shortest number of hops) between nodes \( A \) and \( B \). Pseudo code for the routing algorithm is given in Figure 4.

### 2.3 Gemnet

A \((k,m,p)\) Gemnet consists of \( N = k \times m \) \((k = 1, 2, 3, \ldots, \) and \( m = 1, 2, 3)\) nodes arranged in \( k \) columns of \( m \) nodes each and with the last \((k)-th\) and first columns connected together cylindrically where \( p \) is the in-and-out degree of the nodes. Figure 5 shows a 10 nodes Gemnet configuration. In general, node (row, column) is connected to nodes \((c', \lceil r^{p+i} \rceil \mod m)\) where \( i = 0, 1, 2, \ldots, p-1; \)
\[ c' = (c+1) \mod k \]
\[ r' = r \times p. \]

The diameter (maximum hop distance between any two nodes) is given as: \( \lceil \log_m N \rceil + k-1. \)

![Figure 5: A 10 node \((k=2, m=5, p=2)\) Gemnet topology.](image)

The major advantage of Gemnet compared to Shufflenet and De Bruijn graph is that they are not bound by the network size. Hence, it is scalable, but not perfectly symmetric. It is only symmetric if \( k \) and \( m \) values match the Shufflenets \( p \) and \( k \cdot p \) values.

Find distance (\( D \)) between nodes \((c',r') \) & \((c'',r'')\)
\( D = \left(\lceil c'+c' \rceil \mod m \right) \mod k \)

Define Packet Type (\( S \) or \( M \))

Let Number of alternative shortest paths \( S = \left[ \frac{(p^h - r)}{m} \right] \)

If \( S > 1 \)

Then Packet Type is \( M \)

Else Packet Type is \( S \)

If Packet Type is \( M \)

Then forward packet to any of the available port

Else (if packet type is \( S \)) Routing code
\( (R) = \lceil m \times r' - (r' \times p^h) \rceil \mod m \)

Where, \( h \) is the smallest integer
\( h = (D+ik), \) and \( i = 0, 1, 2, \ldots, \)

Figure 6: Pseudo code for Gemnet routing algorithm.
In Gemnet, routing is simple and adaptive as multiple paths exist. Pseudo code for the routing algorithm is given in Figure 6. Route code $R$ specifies the shortest path from source $(c^s, r^s)$ to destination $(c^d, r^d)$ is expressed as a sequence of $h$ base-$p$ digits, $R = [a_1a_2…a_h]_p$. $i^{th}$ node will route the packet on its $a_i$ outgoing link.

3. Design and Implementation

Figure 7 shows a logical node structure used in the simulations based on a 3X3 switch. Packets are inserted and added dropped via additional add and drop ports.

Based on the deflection and store-and-forward routing strategies [19], two types of node structures have been used in tests, with a one-level buffer and without a buffer. Buffer management is beyond the scope of this paper, therefore it is assumed that the buffer in the system is logical. The following assumptions are also made: one level of buffer is either full or empty. In the event of congestion, packet is stored in the buffer if it is not full. Packet in the buffer is released within the next iteration as it has priority over the packet in the node’s inputs. Note that any node model can be adopted based upon the implementation model and will be explained in the following sections. Also, any suitable buffer management scheme can also be incorporated into the node model [20].

Each of the reviewed networks is implemented in synchronous mode based on the model shown in Figure 9. The model can be used to implement a logical topology without having the need for simulation software. Any programming language can be used to implement logical network topologies based on this model. Here, the C programming language was used to implement topologies as outlined below:

**STEP 1** - Initialisation is the start-up process of the algorithms, where the output ports, buffers and all the variables associated with the node are set to clear.

**STEP 2** - Each node checks whether a packet in its input ports have reached its destination and drops the packet if the current node is that packet’s destination.

**STEP 3** - Every network has a different connectivity pattern where node defines its actual connections that are mapped to its output ports.

**STEP 4** - The node makes the routing decision based on the information contained within the packet header by assigning an output port to which the packet will be forwarded to. This is the “heart” of the simulation process, where different strategies and rules have been used to handle packets that directly affect the performance of the algorithm.

**STEP 5** - Packets are copied from input to output ports. In the case of contention (if there is more than one packet forwarded to the same port), one of the packets is granted the preferred output port and the rest will be deflected or buffered. This step is needed in order to use the output ports mutual exclusively.

**STEP 6** - Node creates a new packet and places it on its empty output port, provided that the output port is available.

**STEP 7** - Packets are copied from the output port of the present node to the input port of the next node based on the node connections defined in **STEP 2**.

**STEP 8** - Current network status is displayed. This information is used to verify the correctness of the algorithm.

---

2 Each outgoing link is numbered individually from 0 to $p$ for each node on Gemnet.
STEPS 5 and 7 can be combined into one by copying packets directly from the input port of one node to the input port of the next node. However, in order to guarantee mutual exclusion, packets must first be copied to the output port before being copied to the input ports of the next nodes. Otherwise, packet copied to the input port overwrites the existing packet that has not been processed, thus resulting in packet losses.

In design and simulation process the following assumptions have been made:

- All nodes in the system have a unique node address,
- All nodes within the system are fully connected,
- Packets are processed synchronously by delaying them at the input ports to ensure they arrive at the node at the same time,
- Packets are delayed while the node is processing the header containing the destination information,
- Nodes are identically the same (i.e. processing power, architecture, routing algorithm, etc.),
- Packet size is fixed,
- There is no packet loss due to congestion as packets are deflected when buffering is not available,
- There is no packet loss due to loss in transmission (it is assumed that a packet restoration algorithm runs in the node),
- Packet in the buffer has priority over the packets in the input ports.
- The existing packet in the network has priority over those entering the network. It is assumed that there is buffering and control mechanism for the new packets.

4. Results

All three topologies were simulated by using the model shown in Figure 9, with deflection and store-and-forward routing. A Random traffic model is used to generate packets based on predefined probability and simulations are performed for 1000 iterations.

Figures 10a and 10b show the average delay versus number of nodes for Shufflenet, Gemnet, and De Bruijn graph for deflection and store-and-forward routing respectively. Average delay is defined as the average number of hops taken for a packet to reach its destination and the average number of times that packet is buffered until it reaches its destination. Here, each hopping and buffering has been considered as a one time slot. For a given number of hops, the average delay is obtained by averaging the number of hops over a range of system loads.

In Figure 10a, Shufflenet and Gemnet display the same characteristics, showing gradual increase in the average delay as the node size increases, hence the diameter of the network increases. Moreover, network accommodates more packets at a given time with larger network sizes, therefore, the chance of having contention increases which results in having longer paths to the destination. On the other hand, De Bruijn graph shows a rapid increase in the delay for the node size 64. Results showed that, a packet is delayed on average more than 70 time slots with 300 nodes. De Bruijn graph is not designed for deflection routing, therefore network itself does not act as a buffer like the other two networks which results on having worse performance. However, in Figure 10b results reflects the impact of the buffer on all of the networks particularly on the De Bruijn graph’s performance when store-and-forward routing is employed. For example, with 200 nodes a packet is delayed 7 time slots in Shufflenet and Gemnet on average and 8 time slots in De Bruijn graph when store-and-forward routing is used. However, this value was around 53 in De Bruijn graph and 10 in other two networks when deflection routing is used.

It can be concluded from the results that a buffer has a significant effect on the De Bruijn graph’s performance compared to the other networks. In fact, Shufflenet and Gemnet are designed for deflection routing where the network itself acts as a buffer, hence the results shows.
5. Conclusions

In this paper a model for implementing logical interconnection topologies has been introduced and based on this model three of the widely used logical network topologies have been implemented for deflection and store-and-forward routing. Results showed that introducing a buffer in the network improved the performance of De Bruijn graph significantly. However, although Gemnet’s and Shufflenet’s results have not improved significantly, they performed better than the De Bruijn graph. Buffer management schemes are currently being incorporated to these networks in the same context. Moreover, WDM based node model and congestion resolution scheme is currently being investigated and implemented.

6. References


3- C. Qiao, M. Yoo, Optical Burst Switching (OBS) - A New Paradigm for an Optical Internet, Journal of High Speed Networks 8 (1) (1999) 69-84.


