2018 年度

法政大学 修士論文

Generalized-Star Crossed Cube における平均パケットレイテンシと最短経路探索アルゴリズム

The Average Packet Latency and Fault Tolerant Routing Algorithm for Generalized-Star Crossed Cube

法政大学大学院 情報科学研究科 情報科学専攻
学籍番号 17T0012
佐藤 智文（さとう ともふみ）
Tomofumi Sato
E-mail: 17t0012@cis.k.hosei.ac.jp

指導教員: 李 亜民 教授

目次

図目次

表目次

Abstract

In the previous research, we proposed a Generalized-Star Crossed Cube (GSCC) interconnection network, which focuses on the cost reduction and flexibility in network size. In this research, we discuss the topological properties of GSCC, examine the average packet latency, and propose a fault tolerant routing algorithm. Average packet latency is the time a packet travels from the source node to the destination node. Multiple nodes send packets simultaneously, and there are conflicts on the paths. The fault tolerant routing algorithm tries to find a routing path in the system where some nodes and links may be faulty. As a result, the average packet latency for GSCC is better than hypercube and $(\boldsymbol{n}, \boldsymbol{k})$-Star Graph when traffic load is low, and the proposed fault tolerant routing algorithm achieves 30 percent better performance than the shortest path routing algorithm.

概要

前回の研究では，我々は Generalised-Star Crossed Cube (GSCC) インターコネクションネットワークを提案し，それにはコスト削減とネットワークサイズに柔軟性があることを示した．今回の研究では，Average Packet Latency による GSCC の位相幾何学的性質と，耐故障経路探索アルゴリズムについて提案する．Average Packet Latency とは送信元ノードから宛先ノードまでのパケット経由にかかる時間を計測したものである．複数のノードを同時に送信するため，経路上で衝突が発生する．耐故障経路探索アルゴリズムはいくつかのノードとリンクが故障したときでも経路探索を発見することができるものである．これらの実験結果では，GSCC における Average Packet Latency ではトラフィック負荷が小さいとき，Hypercube や $(\boldsymbol{n}, \boldsymbol{k})$-Star Graph よりも優れていて，提案した耐故障経路探索アルゴリズムでは最大最大経路探索アルゴリズムより 30 パーセントの到達率向上を達成することに成功した．

1. 導入

近年，世界中で大規模な計算を並列分散で処理できるスーパーコンピュータが AI 分野や Deep Learning など多岐にわたり利用されている． その性能を向上させるために，コンピュータ同士を相互接続する規模は年々増大し続けている．特にコンピュータの性能をランキング形式にした Top500 では，2018 年 11 月現在第 1 位で海外のスーパーコンピュータである Summit は 230 万個を超えるコア数で構成されている．日本でも 2018 年に稼働開始した AI 向けクラウド向け計算システムのスパコンである ABCI (AI Bridging Cloud Infrastructure - AI 橋渡しクラウド) が第 7 位へとランクインして，そのコンピュータの規模は 391,000 個ものコアで構成されている．これらのことから，コンピュータの更なる計算速度向上のためには，コンピュータの数を増加することが必要であるといえる．ここで，並列分散処理をするコンピュータの速度を向上させる方法の 1 つとして，コンピュータ同士の接続方法があげられる．処理速度を高速にするつなぎ方の例として，全てのコンピュータ同士を単純に全接続することである．しかしながら，次数の増大によるリンク数も膨大となり，費用が非常に高くなる問題点がある．一方で，コンピュータ同士をリング状に接続すると費用を抑えることができるが，データ送信する際に経由するコンピュータの数が多くなり処理に時間がかかる問題点がある．これらのことから，これら 2 つの問題点を解消できるグラフ (トポロジ）を設計する必要がある．

以前の研究で新たなトポロジである Generalized-Star Crossed Cube (GSCC(n, k, m)) [1] を提案し，膨大なネットワークサイズでも低次数，小さい直径でかつネットワークサイズに柔軟性があることを示した．今回は，実際にノードの一部である計算ノードからパケットを生成して宛先ノードに送信して実行時間を計測する Average Packet Latency [2] を実装する．そして，既存の研究である Hypercube [3] と Crossed Cube [4]，(n, k)-Star Graph [5]，Generalized-Star Cube [6] と比較し，本トポロジの有意性を検証する．さらに，本研究では，ノードやリンクが故障し使用不可の場合でも迂回して宛先ノードまで経路探索を行える耐故障経路探索アルゴリズムを実装する．正しく宛先ノードへデータを送信できるアルゴリズムを作成し，かつ実行時間をできるだけ抑えることができるアルゴリズムを提案することを目標とする．

2. 関連研究

この節では積グラフ Generalized-Star Crossed Cube の単体のトポロジである 2 つのトポロジと Generalized-Star Cube について説明する．

2.1. Crossed Cube

Crossed Cube とは，Hypercube を変形させたものであり，ノード数による次数は変化しないが，直径を Hypercube が持つ m から $\lceil (m+1)/2 \rceil$ とおよそ半分の大きさとなるものである．アドレスは m 桁の 2 進数で表すことができる．ゆえに Crossed Cube はノード数が 2 の累乗数のサイズのみ設計することができる．2 つのトポロジの異なる部分の例として 3 次元 Hypercube(図 1) と Crossed Cube(図 2) で紹介する．ノードのアドレス番号と次数は同じであるが，中央部分の接続方法が交差されている点が異なる．例えばアドレス 111 の隣接ノードは Hypercube では 110，101，011 であるが Crossed Cube の隣接ノードは 110，101，001 となり 1 つのみ異なる．このノードと隣接ノードとなるアドレスの条件は後ほど紹介する．もう 1 つの例として 4 次元 Crossed Cube は図 3 に示す．3 次元 Crossed Cube と比べて次数と直径が 1 つずつ増え，ノード数は 2 つの 3 次元 Crossed Cube で構成されているため，2 倍となる．ノード数は 16 であり，次数と直径はそれぞれ 4 と 3 である．

次に隣接ノードのアドレス条件について紹介する．例として，8 次元 Crossed Cube のアドレス $\mathrm{CQ}(8) = 01101100$ で最上位ビットを変化させるとき，最下位ビットから変化させるビット未満の 2 ビットずつ値を確認する．ここで，上位ビットを取得できる数が 1 つのみであった場合，ビット反転しないため無視する．その 2 ビットが隣接ノードのアドレスを決定するために必要となる R ($R = (00,00),(10,10),(01,11),(11,01)$) の関係を使用する．この 2 ビットの関係を Crossed Cube のペア関係といい，00 または 10 であった場合変化せず，01 または 11 であった場合上位ビットも反転され，それぞれ 11，01 となる．つまり先程与えられた CQ(8) の最上位ビットを変化させたときの隣接ノードのアドレスは次のようになる．括弧はペア関係を示してあり，この 2 ビットの値によってビット反転するか決定する．下線部は実際に反転させるビット以外に変化した部分を示している．

$$(0(1)(10)(11)(00)) \rightarrow (1(1)(10)(\underline{0}1)(00))$$

また，このトポロジの特徴として，低次数であることによる低コストであることや，他の Hypercube の変形したトポロジよりも経路探索アルゴリズムが容易などがあげられる．

2.2. (n,k)-Star Graph

(n,k)-Star Graph ((n,k)-Star) とは，ノード数が $n!/(n-k)!$ で表すことができるトポロジである．次数は k の値に依存せず常に $n-1$ である．直径は n と k のパラメータで次の 2 つの状態に場合分けされる．($1 \leq k \leq \lfloor n/2 \rfloor$) の場合の $2k-1$ と，($\lfloor n/2 \rfloor + 1 \leq k \leq n-1$) の場合の $k + \lfloor (n+1)/2 \rfloor$ である．次数は k の値では変化しないため，n と k の値次第で

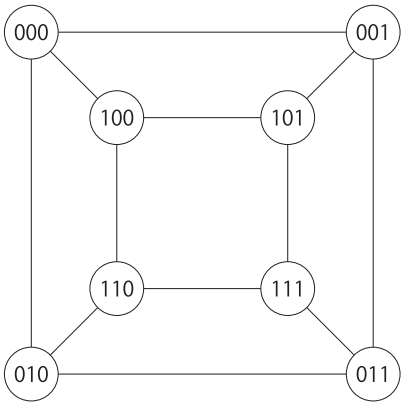

図 1: 3-Hypercube

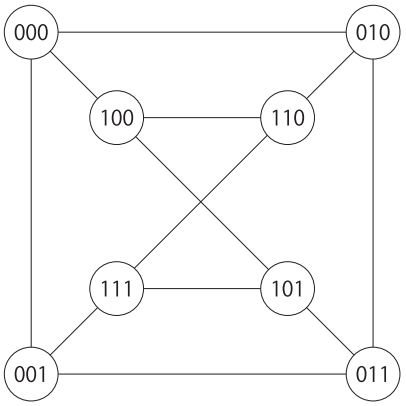

図 2: 3-Crossed Cube

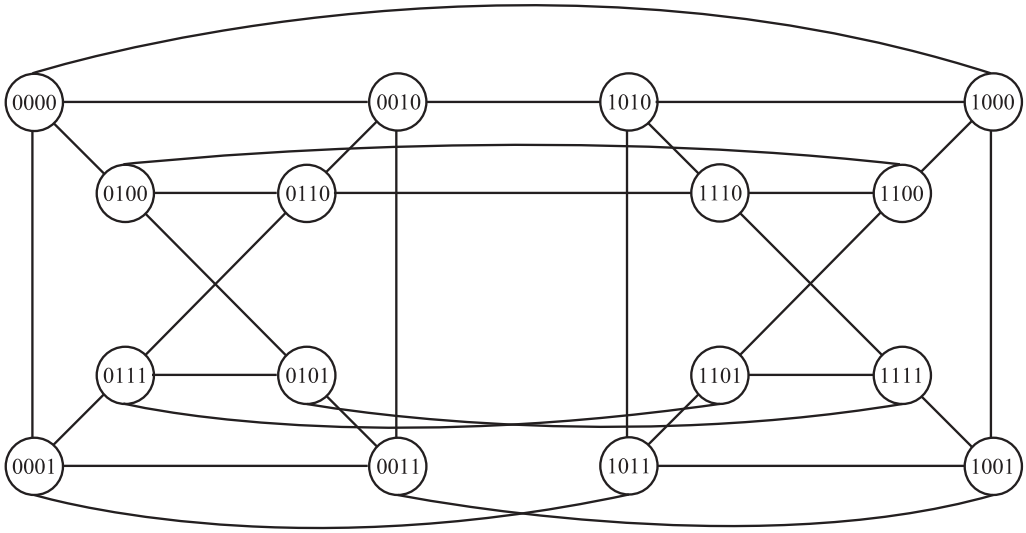

図 3: CQ(4)

は直径とコストが非常に小さくなる．このトポロジは図 4 と図 5 のように表すことができる．図 4 のノード数は 8，次数は 2 で直径は $k + \lfloor (n-1)/2 \rfloor = 3$ であり，図 5 のノード数は 12，次数は 3，直径は $2k - 1 = 3$ となる．特に図 5 では，4 つの 3-完全グラフが 1 つずつ接続されている．また，アドレス番号は 1 から n の数値を k 個並べたものからなる．特に，$k = n - 1$ のときは Star Graph (n-Star) と同型になる．このトポロジの特徴として，低次数，小さい直径，低コスト，左右同型などが挙げられる．

2.3. Generalized-Star Cube

このトポロジは Hypercube と (n, k)-Star Graph の積グラフである．このトポロジは Hypercube の m，(n, k)-Star Graph 部分の n と k の 3 つの変数を所持している．これら 3 つの変数を変えることで様々なサイズを設計することができる．このトポロジのノード数は，2 の累乗数または n-元集合から k-順列を取り出すもののどちらかを満たすことができる場合，このトポロジを設計することができる．Star Cube と比較すると，パラメータ k を用いるこ

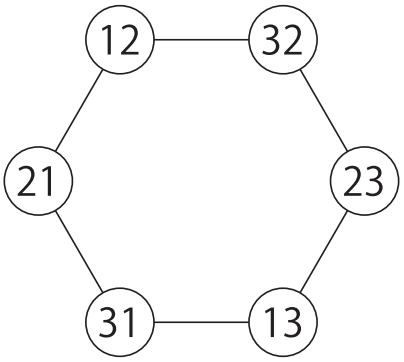

図 4: (3,2)-Star Graph

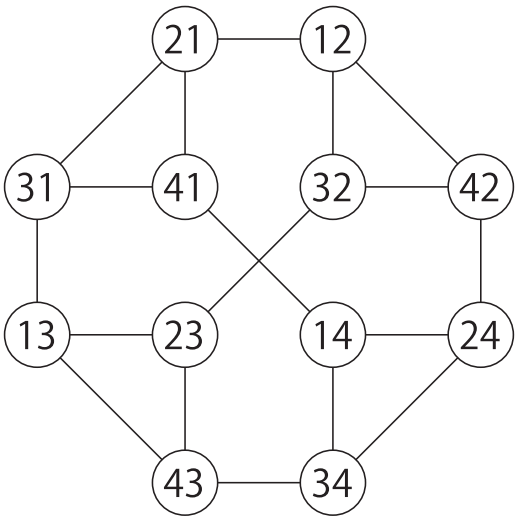

図 5: (4,2)-Star Graph

とができるため GSC(n,k,m) はよりも高い柔軟性を保持している．次数はこれら 2 つの和 $m+n-1$ である．直径は (n,k)-Star Graph と同様に k の値で 2 つの状態に場合分けされる．図 6 は，GSC(3,2,3) のトポロジを示している．(3,2)-Star Graph に 6 つの HQ(3) を埋め込んでつくられたトポロジになる．また，このトポロジは Hypercube に (n,k)-Star Graph を埋め込んで作ることもできる．このトポロジは図 7 で示す．これは，HQ(3) のノード部分に 8 つの (4,2)-Star Graph を埋め込んでつくることができる．このトポロジのアドレスは 2 つのアドレスを持つ．Hypercube 部分が持つ m ビットの 2 進数，(n,k)-Star Graph 部分が持つ 1 から n の数値を k 個並べたものからなる．

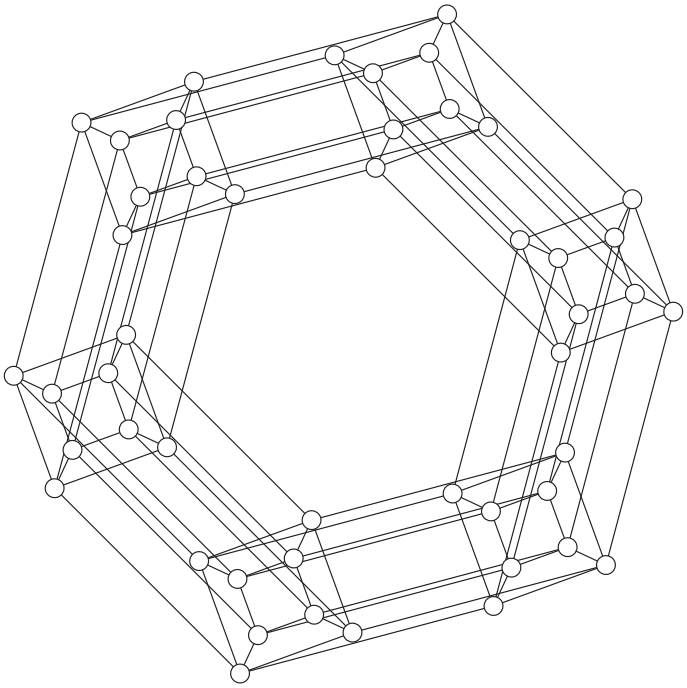

図 6: (3,2)-Star Graph x HQ(3)

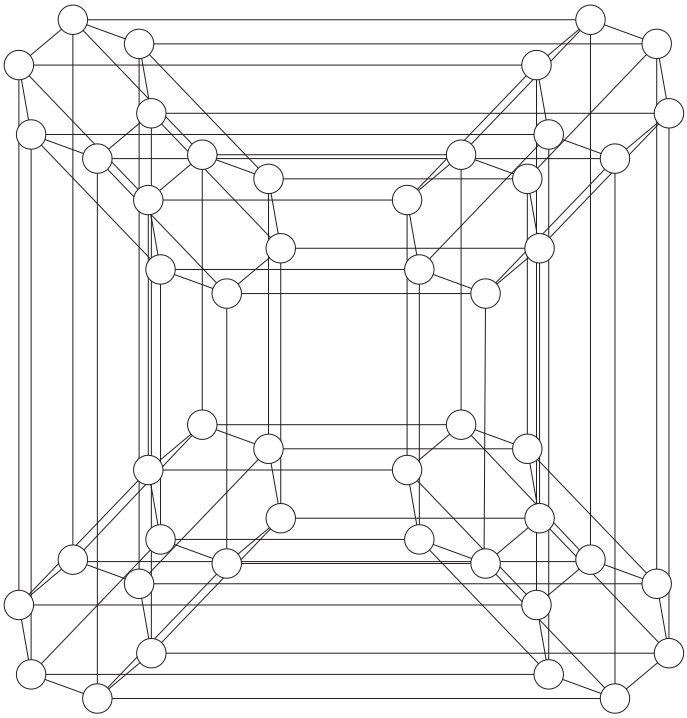

図 7: HQ(3) x (3,2)star

3. Generalized-Star Crossed Cube

この節では、Generalized-Star Crossed Cube の特性と，そのトポロジの最短経路探索アルゴリズムについて述べる．

3.1. Generalized-Star Crossed Cube の性質

Generalized-Star Crossed Cube (GSCC(n, k, m)) は，Crossed Cube と (n, k)-Star Graph の積グラフのことであり，Generalized-Star Cube (GSC(n, k, m)) が持つ Hypercube 部分を Crossed Cube に置き換えたものである．積グラフとは，トポロジのノード部分に他のトポロジを埋め込むことで設計することができる．これによって，ノード数は 2 つのトポロジの積で表し，次数と直径は 2 つのトポロジが持つ次数と直径の和で表現することができる．つまり，パラメータを Crossed Cube が持つパラメータ m と (n, k)-Star Graph が持つ 2 つのパラメータ n, k の 3 つを用いることができるため，ネットワークサイズの柔軟性が非常に高く，かつ Hypercube が持つ大きめな直径を削減することができるため，コストを軽減することができる．ここで，パラメータが $n = k$ の場合，Star-crossed cube [7] と同形になる．図 8 と図 9 はそれぞれ (n, k)-Star Graph に Crossed Cube を埋め込んだもの，Crossed Cube に (n, k)-Star Graph を埋め込んだものとなり，どちらも同じ性質を持っている．例えばノード数と次数，直径はどちらもそれぞれ 48，5，5 である．GSCC のアドレス番号 $\mathrm{GSCC}(n, k, m) = (s, u)$ は CQ(m) が持つ m 桁の 2 進数アドレス ($s = \{s_{m-1}s_{m-2}...s_1s_0\}, s_i \in \{0, 1\}$) と，$(n, k)$-Star Graph が持つ k 桁の 1 から n までの 10 進数アドレス ($u = \{u_0u_1...u_{k-1}\}, u_j \in \{1, 2, ..., n\}$) で表すことができる．このトポロジの特徴として，任意のネットワークのサイズが設計でき柔軟性が高い点や，直径が小さいことによるコストパフォーマンスがよくなる点，最短経路探索アルゴリズムが比較的容易な点などが挙げられる．表 1 では，Crossed Cube と GSC，GSCC などの位相幾何学的性質を示したものである．GSC と比較すると，GSCC は Hypercube が持つ直径 m が Crossed Cube が持つ直径 $\lceil (m + 1)/2 \rceil$ とおよそ半分になることによりその分だけ直径が小さくなる．

3.2. 基本用語と証明

GSCC(n, k, m) の定理について紹介する前に，下記に相互接続ネットワークについての基本用語について説明する．この論文では、相互接続ネットワークは無向グラフとしている．したがって，頂点はプロセッサーに対応し，辺は双方向通信リンクに対応している．

定義 *1.* 相互接続ネットワークは有限グラフ $\boldsymbol{G} = \{\boldsymbol{V}, \boldsymbol{E}\}$ であり，$\boldsymbol{V}$ と $\boldsymbol{E}$ はそれぞれ頂点またはノードの集合，辺またはリンクの集合を表す．

定義 *2.* グラフ $\boldsymbol{G}$ における頂点 v の次数は，辺の数が v 上に示している物に等しい．

定義 *3.* グラフ $\boldsymbol{G}$ の直径 D_G は $\max\{d_G(u, v) | u, v \in \boldsymbol{V}\}$ であり，ここでの d_G は u と v の間の距離である．

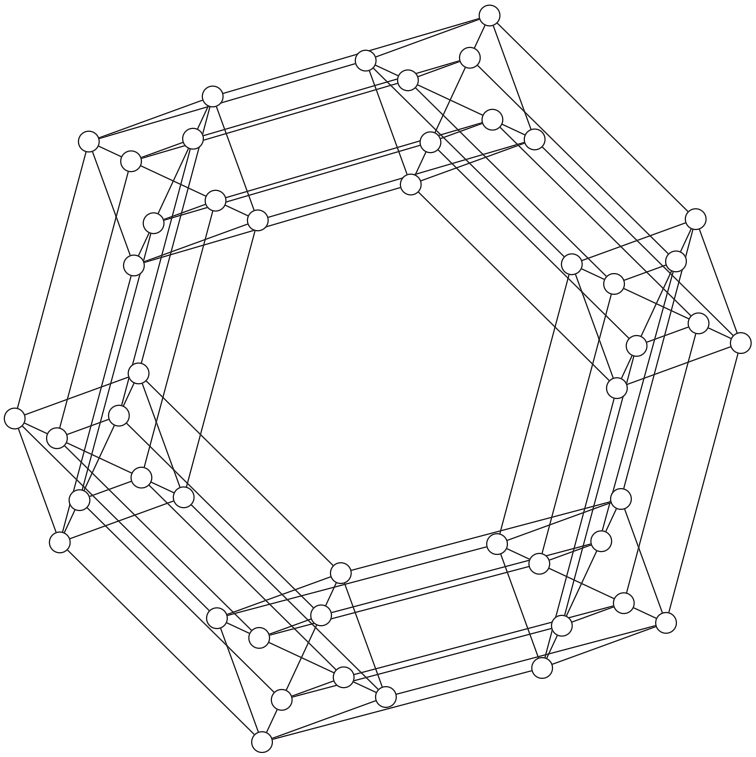

図 8: CQ(3) x NK-Star(3,2)

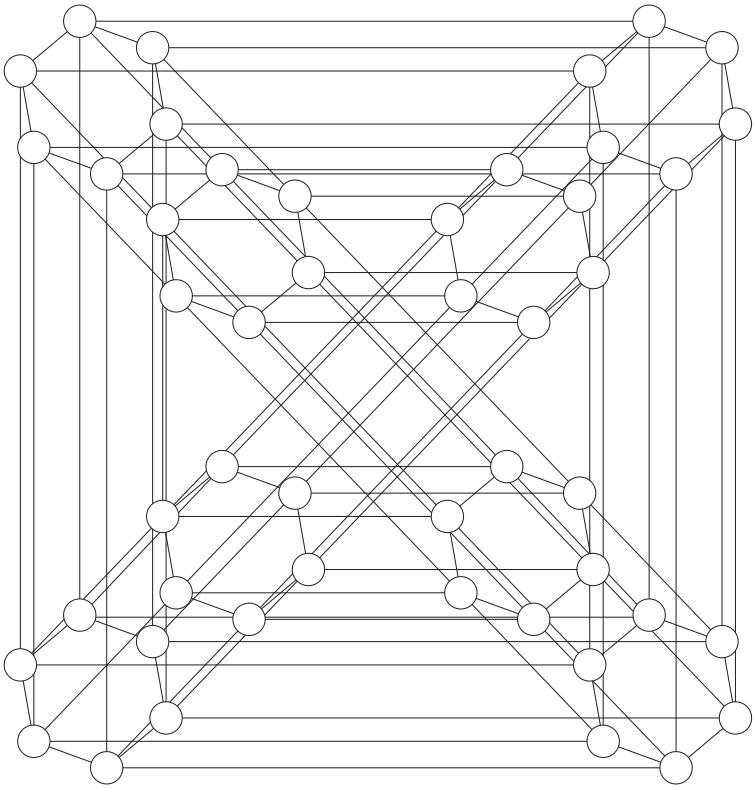

図 9: NK-Star(3,2) x CQ(3)

表 1: トポロジの比較

	HQ(m)	CQ(m)	(n,k)-Star	GSC(n,k,m)	GSCC(n,k,m)
ノード数	2^m	2^m	$\frac{n!}{(n-k)!}$	$2^m\frac{n!}{(n-k)!}$	$2^m\frac{n!}{(n-k)!}$
次数	m	m	$n-1$	$m+n-1$	$m+n-1$
リンク	$m2^{m-1}$	$m2^{m-1}$	$\frac{n!}{(n-k)!}\frac{n-1}{2}$	$2^{m-1}\frac{n!}{(n-k)!}(m+n-1)$	$2^{m-1}\frac{n!}{(n-k)!}(m+n-1)$
直径	m	$\lceil\frac{m+1}{2}\rceil$	$2k-1$ $(1\le k\le\lfloor\frac{n}{2}\rfloor)$ $k+\lfloor\frac{n-1}{2}\rfloor$ $(\lfloor\frac{n}{2}\rfloor+1\le k\le n-1)$	$m+2k-1$ $(1\le k\le\lfloor\frac{n}{2}\rfloor)$ $m+k+\lfloor\frac{n-1}{2}\rfloor$ $(\lfloor\frac{n}{2}\rfloor+1\le k\le n-1)$	$\lceil\frac{m+1}{2}\rceil+2k-1$ $(1\le k\le\lfloor\frac{n}{2}\rfloor)$ $\lceil\frac{m+1}{2}\rceil+k+\lfloor\frac{n-1}{2}\rfloor$ $(\lfloor\frac{n}{2}\rfloor+1\le k\le n-1)$
コスト	m^2	$m\times\lceil\frac{m+1}{2}\rceil$	$(n-1)(2k-1)$ $(1\le k\le\lfloor\frac{n}{2}\rfloor)$ $(n-1)(k+\lfloor\frac{n-1}{2}\rfloor)$ $(\lfloor\frac{n}{2}\rfloor+1\le k\le n-1)$	$(m+n-1)(m+2k-1)$ $(1\le k\le\lfloor\frac{n}{2}\rfloor)$ $(m+n-1)(m+k+\lfloor\frac{n-1}{2}\rfloor)$ $(\lfloor\frac{n}{2}\rfloor+1\le k\le n-1)$	$(m+n-1)(\lceil\frac{m+1}{2}\rceil+2k-1)$ $(1\le k\le\lfloor\frac{n}{2}\rfloor)$ $(m+n-1)(\lceil\frac{m+1}{2}\rceil+k+\lfloor\frac{n-1}{2}\rfloor)$ $(\lfloor\frac{n}{2}\rfloor+1\le k\le n-1)$

定義 *4.* 全ての頂点が同じ次数ならば，それは正規グラフである．

ここからはノード数，直径，次数などの定義をはじめリンク数やコストなどの証明もする．

定理 *1.* GSCC(n,k,m) のノード数は $2^m\times n!/(n-k)!$ である．

証明 *1.* Crossed Cube のノード数は 2^m である．また，(n,k)-Star Graph のノード数は $n!/(n-k)!$ である．GSCC(n,k,m) はこれらの積グラフであるから，ノード数は 2 つのグラフの積 $2^m\times n!(n-k)!$ となる． ■

定理 *2.* GSCC(n,k,m) の直径は $1\le k\le\lfloor n/2\rfloor$ のとき $\lceil(m+1)/2\rceil+2k-1$ で，$\lfloor n/2\rfloor+1\le k\le n-1$ のとき $\lceil(m+1)/2\rceil+k+\lfloor(n-1)/2\rfloor$ である．

証明 *2.* Crossed Cube の直径は $\lceil(m+1)/2\rceil$ である．また，(n,k)-Star Graph の直径は $(1\le k\le\lfloor n/2\rfloor)$ のとき $2k-1$ である．GSCC(n,k,m) はこれらの積グラフであるから，直径は 2 つのグラフの和で表す事ができる．，$(1\le k\le\lfloor n/2\rfloor)$ のとき $\lceil(m+1)/2\rceil+2k-1$ で，$(\lfloor n/2\rfloor+1\le k\le n-1)$ のとき $(\lceil(m+1)/2\rceil+k+\lfloor(n-1)/2\rfloor)$ となる．

■

定理 *3.* GSCC(n,k,m) の次数は $m+n-1$ である．

証明 *3.* Crossed Cube の次数は m である．また，(n,k)-Star Graph の次数は $n-1$ である．GSCC(n,k,m) はこれらの積グラフであるから，次数は 2 つのグラフの和 $m+n-1$ となる． ■

定理 *4.* GSCC(n,k,m) のリンク数の合計は $2^{m-1}n!/(n-k)!(m+k+1)$ である．

証明 *4.* GSCC(n,k,m) は，(n,k)-Star Graph の型に $k!/(n-k)!$ 個の Hypercube が埋め込まれていると考える．したがって，Crossed Cube 部分のリンク数の合計は $(m2^m)/2 \times n!/(n-k)! = m2^{m-1}n!/(n-k)!$ となる．$(m2^m)/2$ は 2^m の各ノードが m 個隣接しているが，全ての辺が 2 倍に数えられているので，2 で割る．次に各 Crossed Cube 部分は $n-1$ 個隣にリンクされていて，それは (n,k)-Star Graph 部分を通して同じ Crossed Cube 部分と繋がっている．したがって，(n,k)-Star Graph の辺は $(n-1)2^m/2 \times n!/(n-k)! = (n-1)2^{m-1}n!/(n-k)!$ となる．ゆえに，GSCC(n,k,m) のリンク数は合計で $m2^{m-1}n!/(n-k)! + (n-1)2^{m-1}n1(n-k)! = 2^{m-1}n!/(m-k)!(m+n-1)$ となる．■

定理 *5.* GSCC(n,k,m) のコストは $(1 \le k \le \lfloor n/2 \rfloor)$ のとき $(m+n-1)(\lceil (m+1)/2 \rceil + 2k+1)$ で、$(\lfloor n/2 \rfloor + 1 \le k \le n-1)$ のとき $(m+n-1)(\lceil (m+1)/2 \rceil + 2k-1)$ である．

証明 *5.* ネットワークのコストは コスト = 次数 × 直径 である．定理 3 から，GSCC(n,k,m) の次数は $(m+n-1)$ である．また，定理 2 から，直径は $(1 \le k \ne \lfloor n/2 \rfloor)$ のとき $(m+n-1)(\lceil (m+1)/2 \rceil + n-1)$ で，$(\lfloor n/2 \rfloor + 1 \le k \le n-1)$ のとき $(\lceil (m+1)/2 \rceil + k + \lfloor (n-1)/2 \rfloor)$ である．従って GSCC(n,k,m) のコストは，$(1 \le k \le \lfloor n/2 \rfloor)$ のとき $(m+n-1)(\lfloor (m+1)/2 \rfloor + 2k-1)$ で，$(\lfloor n/2 \rfloor + 1 \le k \le n-1)$ のとき $(m+n-1)(\lceil (m+1)/2 \rceil + k + \lfloor (n-1)/2 \rfloor)$ となる．■

定理 *6.* GSCC(n,k,m) は正規グラフである．

証明 *6.* Crossed Cube と (n,k)-Star Graph は正規グラフである．ゆえに GSCC(n,k,m) は積グラフであり，定義 4 から，GSCC(n,k,m) は正規グラフである．■

3.3. 最短経路探索アルゴリズム

この小節では，パケット送信などに使われる経路探索アルゴリズムについて述べる．最短経路探索方法は，次のように構成される．

1) Crossed Cube 部分の送信元ノード s を宛先ノード t へ移動する．
2) (n,k)-Star Graph 部分の送信元ノード u を宛先ノード v へ移動する．

最短経路アルゴリズムは次の Algorithm $1 \sim 4$ からなる．このアルゴリズムは Crossed Cube 部分を探索したあと，(n,k)-Star Graph 部分を探索するものになる．Algorithm 1 は Crossed Cube 部分の最短経路探索アルゴリズム [8] である．ここで，最短経路の条件として ρ を定義する．ρ はペア関係の選択される回数を表し，数値分回選択される．ρ を定義する前に i^* を定義する．まず，l を s と t で異なる最大ビットとする．次に $i^* = \lfloor l/2 \rfloor$ と定義する．これは，Crossed Cube の隣接ノードによるビットの変化が 2 ビットずつで決定するために必要になる．$j \ge i^*$ のとき，ρ と s,t の関係は式 (1)，式 (2) のようになる．i^* より大きい場合，ビット反転する必要がないため 0 となる．また，$j = i^*$ のときはペア関係でないため両方のビットが違う場合 ρ は 2 となり，片方のみのビットが異なる場合 1 となる．

$$\rho_{i^*}(s,t) = 0 \quad (j \ge i^* + 1) \tag{1}$$

$$\rho_j(s,t) = \begin{cases} 2 & (s_{i^*+1}s_{i^*} = \overline{t}_{i^*+1}\overline{t}_{i^*}) \\ 1 & (Otherwise) \end{cases} \tag{2}$$

Algorithm 1 CQ_Part(s,t)

Input: CQ(m) of Source node s
Input: CQ(m) of Destination node t
Output: Address number of s
  define $\rho_i(s,t)$ for all $0 \le i \le \lfloor \frac{m}{2} \rfloor$,
  $Q \leftarrow \{j | \rho_j(s,t) \neq 0\}$,
  $T \leftarrow \{\rho_j(s,t) \neq 0, j < i^*$ and
    $\overline{s}_{2j+1}s_{2j} \overset{\text{d.-p.}}{\sim} t_{2j+1}t_{2j}$ or $s_{2j+1}\overline{s}_{2j} \overset{\text{d.-p.}}{\sim} t_{2j+1}t_{2j}\}$
  if $T \neq \phi$ **then** /***** Step1 *****/
    find a $j \in T$ and call *ONE_STEP_ROUTE*(i,Q)
  else if $Q \neq \phi$ **then** /***** Step2 *****/
    if either $\overline{s}_{2i+1}s_{2i} \overset{\text{d.-p.}}{\sim} t_{2i+1}t_{2i}$ or $s_{2i+1}\overline{s}_{2i} \overset{\text{d.-p.}}{\sim} t_{2i+1}t_{2i}$ holds for some $i \in Q$ **then**
      choose such smallest i
    else
      $i \leftarrow max\{j | j \in Q\}$
    end if
    Call *ONE_STEP_ROUTE*(i,Q)
  end if
  return s

Algorithm 2 ONE_STEP_ROUTE_(j,Q)

  if $\rho_j(s,t) = 2$ **then**
    route to s', i.e., the $2j$th or $(2j+1)$th neighbor of s
    $\rho_j(s,t) \leftarrow \rho_j(s,t) - 1$;
  else
    if $\overline{s}_{2j+1}s_{2j} \overset{\text{d.-p.}}{\sim} t_{2j+1}t_{2j}$ **then**
      route to s' the $(2j+1)$th of s
    else if $s_{2j+1}\overline{s}_{2j} \overset{\text{d.-p.}}{\sim} t_{2j+1}t_{2j}$ **then**
      route to s' the $2j$th of s
    end if
    $\rho_j \leftarrow \rho_j(s,t) - 1$
  end if
  $s \leftarrow s'$
  return s

次に，$j < i^*$ のときを考える．Crossed Cube は上記の通り上位ビットを変化すると下位ビットもペア関係により変化する．そのために，下の 3 つ条件のいずれかを満たす s と t を距離維持ペア関係 (distance-preserve pair related) が成り立つとして，その式を $s_{2j+1}s_{2j} \overset{\text{d.-p.}}{\sim} t_{2j+1}t_{2j}$ のように表す．これにより距離維持ペア関係が成り立つ部分のビット部分を変化させないようにする．また，$j < i^*$ のときの ρ と s,t の関係は式 (3) のようになる．

1) $(s_{2j+1}s_{2j}, t_{2j+1}t_{2j}) \in \{(01,01),(11,11)\}$ かつ $\sum_{i=j+1}^{\lfloor \frac{m-1}{2} \rfloor} \rho_i(s,t)$ が偶数

2) $(s_{2j+1}s_{2j}, t_{2j+1}t_{2j}) \in \{(01,11),(11,01)\}$ かつ $\sum_{i=j+1}^{\lfloor \frac{m-1}{2} \rfloor} \rho_i(s,t)$ が奇数

Algorithm 3 NKStar_Part(u, v)

Input: (n, k)-Star of Source node u
Input: (n, k)-Star of Destination node v
  if $u_0 = v_0$ **then**
    find minimum j that satisfies $u_j \neq v_j$
    swap u_0 for u_j
  end if
  if $u_0 = v_j$ **then**
    swap u_0 for u_j
  else
    find min. value min of v that satisfies $u \not\subset v$
    $u_0 \leftarrow min$
  end if
  $S \leftarrow S + \{u\}$
  return u

Algorithm 4 Shortest Path Routing Algorithm

Input: CQ(m), (n, k)-Star of Source node s, u
Input: CQ(m), (n, k)-Star of Destination node t, v
Output: Routing order of S
  while $s \neq t$ or $u \neq v$ **do**
    if $s \neq t$ **then**
      $p \leftarrow$ CQ_Part(s, t)
    else if $u \neq v$ **then**
      $p \leftarrow$ nkStar_Part(u, v)
    end if
    $S \leftarrow S + p$
  end while
  return S

3) $(s_{2j+1}s_{2j}, t_{2j+1}t_{2j}) \in \{(00, 00), (10, 10)\}$

$$\rho_j(s,t) = \begin{cases} 0 & (s_{2j+1}s_{2j} \overset{\text{d.-p.}}{\sim} t_{2j+1}t_{2j}) \\ 1 & (Otherwise) \end{cases} \tag{3}$$

Algorithm 2 は Crossed Cube 部分の経路探索する際に用いられる．ここでペア関係の上位ビットまたは下位ビットを選択し経路探索するものとなる．

また，(n, k)-Star Graph 部分の隣接ノードは，先頭のアドレスだけが異なる，あるいは先頭と先頭を除いたアドレスの長さの 1 つにあるものが交換されたものである．経路探索の流れとして，先頭のアドレスが異なる場合，それが宛先ノードのアドレスにある場合それと交換する．宛先ノードのアドレスに無い場合は，宛先ノードに存在して送信元ノードにない最小の値を入れる．またアドレスの先頭が同じであるが他のアドレス番号が異なる場合，先頭とその異なる部分の中から最も左にあるものと入れ替える．これらのことををまとめた (n, k)-Star Graph 部分の最短経路アルゴリズムは次の Algorithm 3 に示す．また，2 つの経路を合わせた Generalized-Star Crossed Cube の探索アルゴリズムを Algorithm 4 に示す．

ここで，最短経路の例として，パラメータ n, k, m をそれぞれ $8, 5, 8$ と設定する．Crossed Cube 部分の送信元ノード $s = 01001001$ と宛先ノード $t = 10010001$，(n, k)-Star Graph 部分の送信元ノード $u = 17482$ と宛先ノード $v = 12345$ とする．Algorithm 4 から Crossed Cube 部分の探索から実行する．はじめに ρ と T，Q を定義する．式 1，2，3 より，$\rho(s, t) = \{0, 1, 1, 2\}$ Algorithm 1 から $T = \{2\}$，$Q = \{1, 2, 3\}$ が導かれる．次に Step1 では T の要素があるときはその要素部分を先に探索するものである．ゆえに ONE_STEP_ROUTE$(2, T)$ を実行する．ONE_STEP_ROUTE (Algorithm 2) は探索部分である．$\rho_2(s, t) = 1$ のため，次の距離関係維持関係に送信元ノードにビット反転を片方のみ加えた条件を満たす．今回は $(s_5 s_4, t_5 t_4) = (00, 01)$ かつ $\sum_{i=3}^{3} \rho_i(s, t) = 2$ であるため，$s_5 \bar{s}_4 \overset{\text{d.-p.}}{\sim} t_5 t_4$ が成立する．よって 2j 番目の探索となり s_4 のビット反転を実行し，その結果，$s' = 01011011$ となる．$T = \phi$ となり，Step2 へ進む．Step2 では Q にある要素の中から一度探索すると距離維持ペア関係が成立しその部分の探索なしに宛先ノードへ到達することができるものを優先する．それが成立しない場合，Q にある要素から一番大きい値を選択する．ここで，$Q = \{1, 3\}$ であり，$Q = 1$ のとき $(s_3 s_2, t_3 t_2) = (10, 00)$ かつ $\sum_{i=2}^{3} \rho_i(s, t) = 2$ であるため距離維持ペア関係の条件 3 から s_3 をビット反転したものが成り立つため，$s_3 \bar{s}_2 \overset{\text{d.-p.}}{\sim} t_3 t_2$ が成立する．その後，ONE_STEP_ROUTE の探索部分で s_3 のビット反転をした結果 $s' = 01010001$ となる．

残りは $Q = \{3\}$ かつ $\rho_3(s, t) = 2$ となるため ONE_STEP_ROUTE では 6 番目と 7 番目の探索を順不同で実行することができる．ここでは先に 7 番目の探索，すなわち s_7 のビット反転の実行後 6 番目の探索，s_6 のビット反転をした結果は $s' = 10010001$ となる．これで $s = t$ かつ $Q = \phi$ となり，Crossed Cube 部分の探索は終了する．Crossed Cube 部分のアドレスの探索順序は次からなり，探索回数は 4 である．また，下線部分は前の状態と比較して変化している部分で二重下線部分は探索した部分となる．

$$01001001 \rightarrow 010\underline{\underline{1}}10\underline{\underline{1}}1 \rightarrow 0101\underline{\underline{0}}0\underline{\underline{1}} \rightarrow \underline{\underline{1}}1110\underline{\underline{0}}1 \rightarrow 1\underline{\underline{0}}0100\underline{\underline{1}}$$

次に (n, k)-Star Graph 部分の探索をする．はじめに (n, k)-Star Graph は先頭のアドレスの値を確認する．$u_0(1) = v_0(1)$ であるため送信元ノードと宛先ノードのアドレス番号が正しくない最小の値を探す．今回は左から 2 番目の値 $(u_1 = 7, v_1 = 2)$ が同じではないため，u_0 と u_1 を交換して，$u' = 71482$ となる．次は $u_0 = 7$ で宛先ノードに含まれていないため，送信元ノードにはなく，宛先ノードに含まれている最小の値 (3) を u_0 へ代入する．3 回目の探索では，先頭のアドレスが同じではなくかつ $u_0(4) = v_2$ であるため先頭のアドレス 3 と左から 3 番目のアドレス 4 を交換する．4 回目の探索では，3 回目の探索と同様に先頭のアドレスが同じではなくかつ $u_0(4) = v_2$ であるため先頭のアドレス 3 と左から 3 番目のアドレス 4 を交換する．5 回目の探索は 2 回目の探索と同様に送信元ノードになく，宛先ノードに含まれている最小の値 (5) を u_0 へ代入する．6 回目と 7 回目の探索はそれぞれ $u_0(5)$ と $u_4(2)$ の交換をしたあと，$u_0(2)$ と $u_1(1)$ の交換をする．これで $u = v(12345)$ となるため，(n, k)-Star Graph 部分の探索は終了する．下記には (n, k)-Star Graph 部分で探索した経路である．

$$17482 \to 71482 \to 31482 \to 41382 \to 81342 \to 51342 \to 21345 \to 12345$$

(n,k)-Star Graph 部分の探索回数は 7 となり，これらの積グラフである Generalized-Star Crossed Cube はこれら 2 つ探索の和で表現できるため総合探索回数は $4+7=11$ である．

このアルゴリズムの実行時間は Crossed Cube 部分の探索は $\boldsymbol{O}(m)$ で探索することができる．また，(n,k)-Star Graph 部分の探索は $\boldsymbol{O}(k)$ で探索することができる．ゆえに Generalize-Star Crossed Cube の探索はこれら 2 つの積グラフであるため，これら 2 つの和 $\boldsymbol{O}(m+k)$ となり，Crossed Cube 部分の探索が (n,k)-Star Graph 部分の探索より十分に大きくなる場合 $\boldsymbol{O}(m)$ と表現でき，逆に十分に小さくなる場合 $\boldsymbol{O}(k)$ と表現できる．

4. Average Packet Latency

この節では，Average Packet Latency について説明し，それの実験方法について述べる．

4.1. Average Packet Latency について

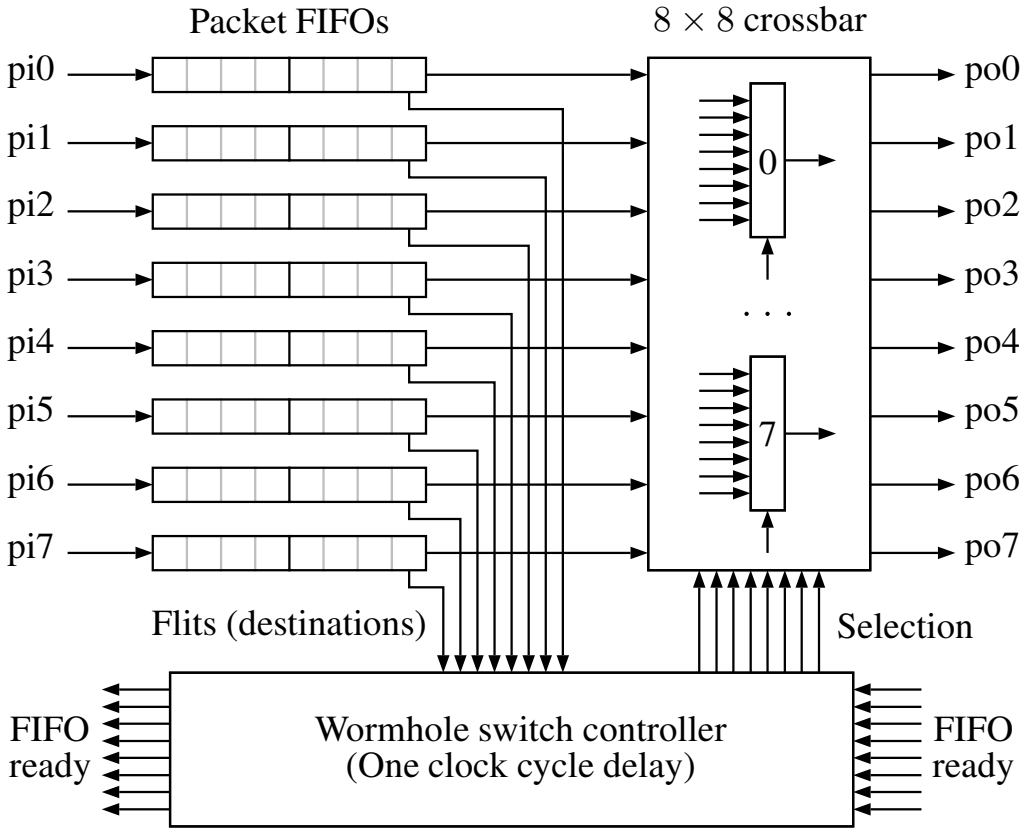

図 10: Router Block Diagram

各ルータに図 10 のような隣接ノードから送信されたパケットを管理し，隣接ノードにパケットを送信するためのシステムを導入する．他ノードの計算ノードから作られたパケットが複数の隣接ノードから送信され，入力として Pi から Packet FIFO に入れられる。Packet FIFO はリンクの個数分と 1 つの計算ノードからパケットを入れるものがあり，1 つのリンクごとに 1 つの FIFO で管理される．一定時間ごとにすべての FIFO から 1 つ宛先ノードなどをの情報を Switch Controller で処理させる．FIFO の中身が空ならば何も処理しないようにする．Switch controller は FIFO の先頭のパケットの宛先ノードを確認して，送信先には経路探索アルゴリズムを用いて次に送信する隣接ノードを決定する．同じ隣接ノードに送信したいパケットが複数存在した場合最も長く FIFO に残っているものを選択して (LRU)、ここで選択されないものはそのまま FIFO 内に残す．そこで決定した FIFO の番号を Crossbar へ送信する．ここで隣接ノードの FIFO が満杯などによる入力が不可能な場合、FIFO ready 信号の Switch controller で Crossbar 内でパケットを送信できないように処理する。Crossbar

はSwitch controllerから送られてきた信号からパケットをどの隣接ノードに送信するか1つずつ選択する．候補がない場合または隣接ノードのFIFOが満杯の場合は何も送信しない．またパケットの生成方法として，一部のノードに計算ノードが1つ付属してあると想定し，その部分から一定時間ごとに1つずつPacket FIFOに入力される．そのパケットをCrossbarで他のノードに送信することでパケットの送受信ができるものになる．ここでもFIFOが満杯の場合はパケットを生成しないようにする．

以上のことから図10の各ブロック図の役割についてまとめると以下のようになる．

- pi0 ～ pi7， po0 ～ po7: 隣接ノードにパケットを送受信させる．
- Packet FIFOs: パケットを一時的に保持しておくもの．crossbarで選択されたときデータの消去を行う．
- 8x8 crossbar: 各FIFOから1つ選択し，poに送信する．
- FIFO ready(入力): 隣接ノードのFIFO状態を確認する信号．
- FIFO ready(出力): 自身のノードのFIFO状態を確認する信号．

4.2. Average Packet Latencyの実験方法

はじめに，全ノードからパケット送信する計算ノードの割合であるトラフィック負荷 $(0.0 < \lambda \leq 1.0)$ を設定する．このトラフィック負荷 λ を0.05刻みで1まで増大させて，実行時間の変化を検証する．計算ノードは，宛先ノードを自身のアドレス以外を指定し，パケットを生成すると同時にパケットを隣接ノードへ送信する際に実行時間を増加させるとする．ここで，パケットが宛先ノードに到着した際に，送信元ノードの到着カウンタを1つ増やす．これらの操作を繰り返し，パケットを送信している送信元ノードの到着カウンタが一定数を超えたときの実行時間を測定する．今回はすべての到着カウンタが200を達成したときの実行時間を計測した．この実験を100回繰り返し、その実行時間の平均値を結果とした。今回は5種類のトポロジを用いて実験を行った．トポロジの種類やパラメータなど次の表2からなる．これの実験結果は次の4.3節となる．また，同じ隣接ノードに送信したいパケットが複数存在して衝突した回数を示したものを4.4節で紹介する．ここで，今回使用するアルゴリズムとして，最短経路アルゴリズム (Algorithm 1から4) を用いる．λ を増大させることによる実行時間の変化を調べる．ここではPacket FIFOの容量を2と8に設定して，実行時間にどのような差が生じるか確認する他にも様々な実験を行い，FIFOの選択方法をLRUからランダムに変えたときの実行結果を4.5節に，宛先ノードを任意ノードから最長距離の1つのノードに変化したときの実行時間については4.6節に示す．

4.3. Average Packet Latencyの結果と考察

この小節では先行研究とトポロジ単体でのトポロジ５つでトラフィック負荷による実行時間の変化について比較する．この実験では，同じノード数にすることは不可能であるため，ノード数をなるべく近づけて実験した．図11と図12では，表2からおよそノード数2,000個のときのトラフィック負荷 λ の増加による実行時間の変化を示している．図11のFIFO容量が2のときのHypercubeとCrossed Cubeを比較すると，λ が小さいときは直径が小さ

表 2: トポロジのパラメータ設定

トポロジ	HQ(11)	CQ(11)	(8,4)-Star	GSC(5,3,5)	GSCC(5,3,5)
ノード数	2048	2048	1680	1920	1920
次数	11	11	7	10	10
直径	11	6	7	10	8

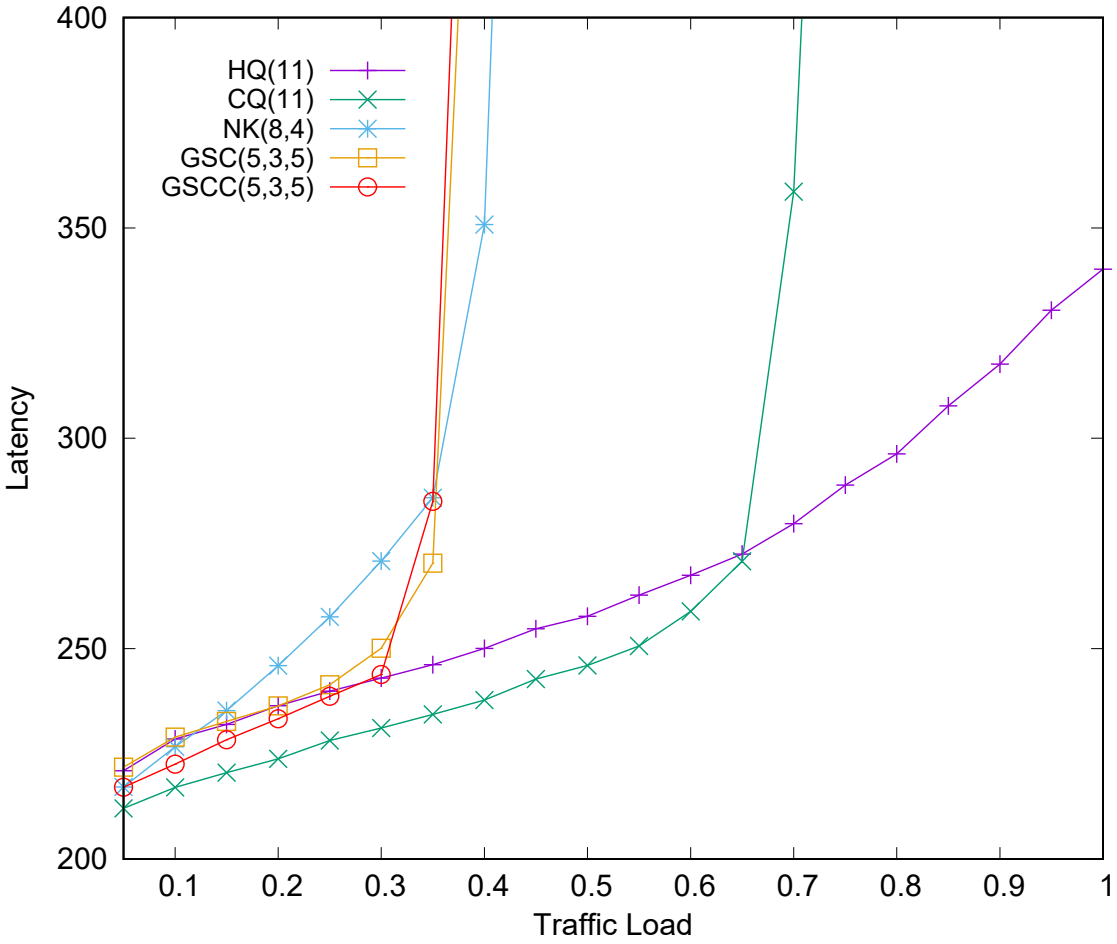

図 11: FIFO 容量が 2 のときの Average Packet Latency

いことでホップ数が少なくなるため、Crossed Cube が実行時間が最大で 12 小さくなっている。しかしながら，λ が 0.65 を超えたあたりから Crossed Cube の傾きが非常に大きくなり，Hypercube のほうが実行時間が小さくなる．これは，最短経路を探索するため，パケットが同じ隣接ノードを参照してパケット同士が詰まるためであると考えられる．(n,k)-Star Graph と Generalized-Star Cube，Generalized-Star Crossed Cube を比較すると，(n,k)-Star Graph は λ が小さいときでも傾きが大きいが，λ が 0.4 より大きくなると GSC と GSCC よりも実行時間が小さくなる．GSCC は λ が 0.25 までは Hypercube，GSC よりも小さくなるが、0.3 で Hypercube より大きくなり、0.35 以上になると GSC より大きくなることが確認できる．

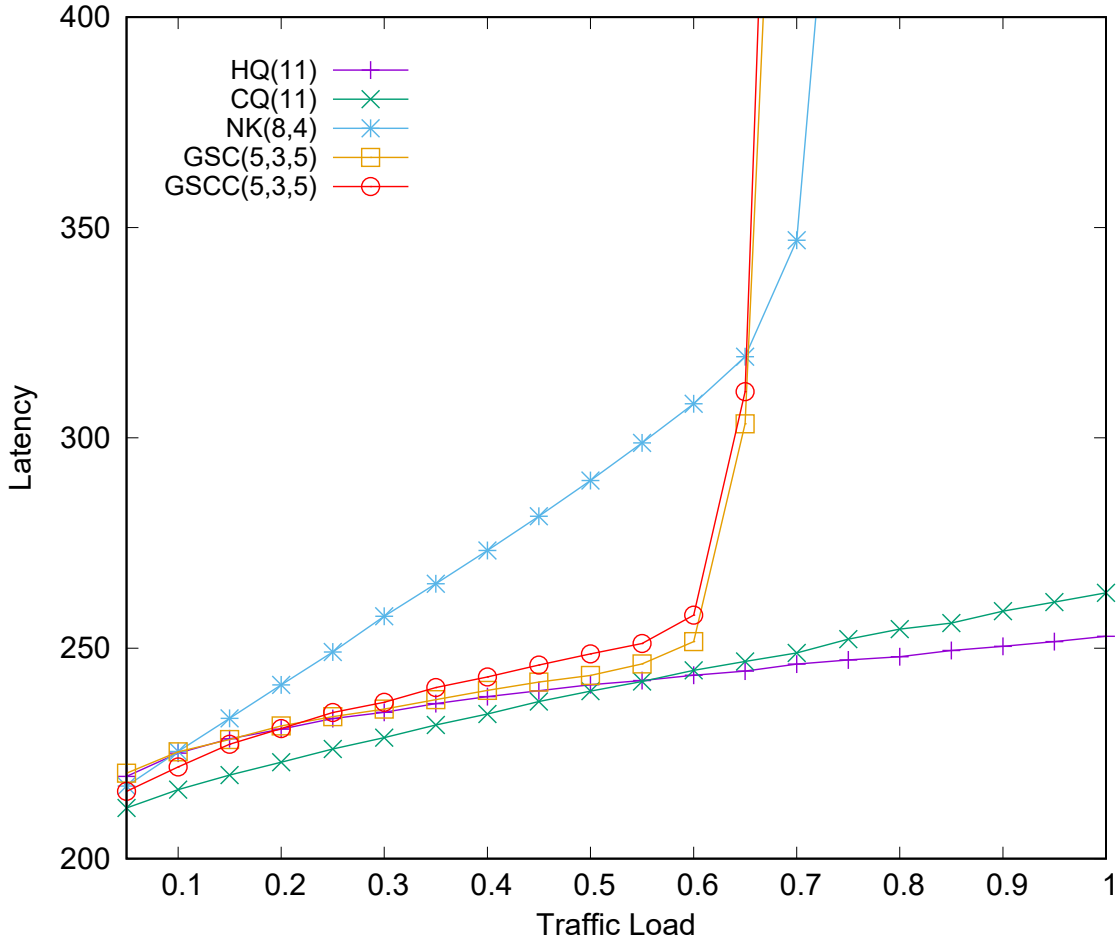

図 12: FIFO 容量が 8 のときの Average Packet Latency

これらのことからトラフィック負荷が小さいとき実行時間が他トポロジより小さくなるため GSCC が優れていることと判断できる．また，GSC と GSC が (n, k)-Star Graph 単体と比較して傾きが急上昇する λ が小さく，かつ傾きがより急である理由として，(n, k)-Star Graph 側の実行時間が Hypercube と Crossed Cube よりも大きく，(n, k)-Star Graph 部分を探索するパケットが多くなった結果パケット同士が詰まってしまい，Hypercube と Crossed Cube 側にも影響を及ぼしているためであると考えられる．

次に図 12 で比較を行う。これは FIFO 容量が 8 のときのそれぞれの Average Packet Latency の実験を行ったものである。Hypercube と Crossed Cube で比較を行うと，どちらも傾きが一定であることが確認できる．しかしながら Crossed Cube がより傾きが大きいため、λ が 0.6 以上になると Hypercube より実行時間が長くなることが確認できる．ここで，他の 3 つのトポロジで比較すると、(n, k)-Star Graph は λ が 0.65 までは実行時間が最も大きくなり，その後は GSC と GSCC が大きくなる．GSCC は λ が 0.2 までは GSC と Hypercube よりも実行時間が小さくなり、その後は 0.55 までは Hypercube と同様に傾きが一定である．しかしながら，λ が 0.6 を超えると傾きが急激に大きくなり、実行時間が指数関数的に大きくなることが確認できる．これも FIFO の容量が 2 の時と同様に (n, k)-Star Graph 側の実行時間が Hypercube と Crossed Cube よりも実行時間がかかるためであると考えられる．

これら 2 つのことから、トラフィック負荷が小さいときは GSCC が Hypercube や (n,k)-Star Graph，GSC よりも優れているが，負荷が大きくなると実行時間が指数関数的に大きくなり，最終的には 5 つのトポロジの中で実行時間が最大となるためあまり適していないことが判明した。また，FIFO 容量を大きくした場合でも実行時間が指数関数的に増加するトラフィック負荷が 0.6 までと FIFO 容量が 2 のときの 0.35 よりも大きくなるのみであり，それ以上のときの増加量はトポロジによる実行時間の順番に変化がないため FIFO 容量による影響するものではないと判明した。

4.4. トラフィック負荷と衝突回数

この小節では，トラフィック負荷 λ の増加による衝突回数の変化について紹介する．ここでは同じ隣接ノードに送信したいパケットが複数個あった場合，その個数分増加させる．1 つまたは 1 つも存在しない場合増加しない．例えば，同じ隣接ノードの送信先に送信するパケットの候補が 2 つであった場合 2 増加して，3 つ存在する場合は 3 増加する．その結果は次の図 13 と 14 からなる．

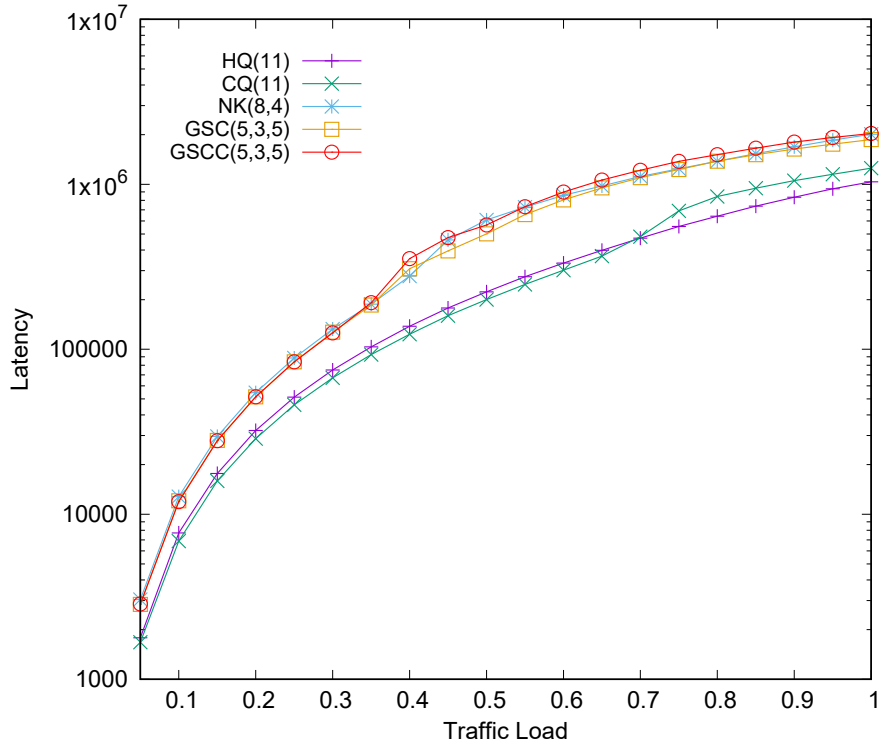

図 13: FIFO 容量が 2 のときの衝突回数

図 13 は FIFO 容量が 2 のときの衝突回数である．Hypercube と Crossed Cube が近い値を示していて，(n,k)-Star Graph と Generalized-Star Cube，Generalized-Star Crossed Cube も近い値を推移している．Hypercube と Crossed Cube を比較するとはじめは λ が 0.6 までは Crossed Cube 側が衝突回数が少ないが，0.7 から 0.75 にかけて急上昇してから再び緩やかな曲線になっていることが確認できる．これは，図 11 から，Crossed Cube の実行時間が急

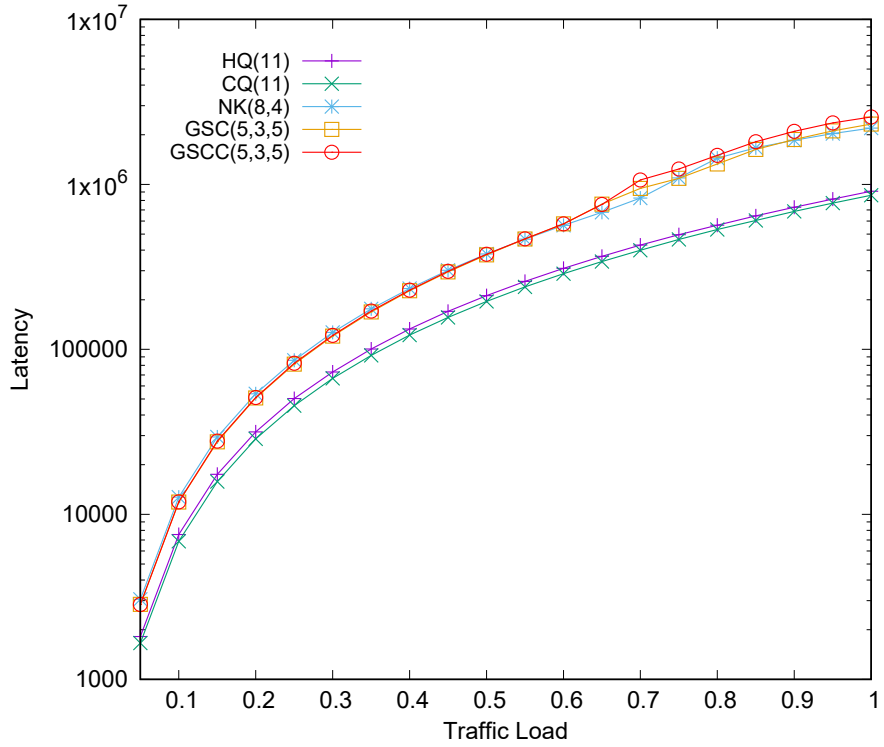

図 14: FIFO 容量が 8 のときの衝突回数

に大きくなるときのトラフィック負荷と同じ部分であることから，衝突回数が増加すると実行時間にも影響があることが判明した．また，他の 3 つのトポロジと比較すると，λ が 0.3 までは (n, k)-Star Graph が最も衝突回数が大きくなる．しかしながら，λ が 0.75 以上になると Generalized-Star Cube が最も少なくなり Generalized-Star Crossed Cube が最も大きくなる．これは，λ の増大による実行時間の関係と全く同じである．

次に図 12 は Packet 容量が 8 のときの衝突回数を示したものである．FIFO 容量が 2 のときと比較すると全体的に 1 割衝突回数が減少している．Hypercube と Crossed Cube と比較すると，図 13 常に Crossed Cube の衝突回数が Hypercube より少なくなる．これは図 12 では λ が 0.6 以上のときに実行時間が Hypercube よりも大きくなるが衝突回数は急に大きくなっていない．このことから，Crossed Cube は直径が小さい分衝突回数が少なくなると判断できる．他の 3 つのトポロジで比較すると，λ が 0.6 までは (n, k)-Star Graph は 2 つのトポロジよりも衝突回数が多いが，λ が 0.65 のときに Generalized-Star Crossed Cube が最も多くなる．これは，FIFO 容量が 2 のときと同様に傾きが急激に大きくなったときに衝突回数がさらに上昇している．これらの実験結果から，実行時間が急激に上昇しない限りは直径が小さくなった分衝突回数を少なくすることができる．

4.5. 選択方法変更による結果と考察

この小節ではパケット同士が衝突した際，パケットの選択方法を LRU とランダムで実行時間の変化について紹介する．この実験ではパケット容量を 2 にしたときの実行時間を調べた．図 15 は実行時間をパケット選択方法をランダムと LRU で比較したものである．

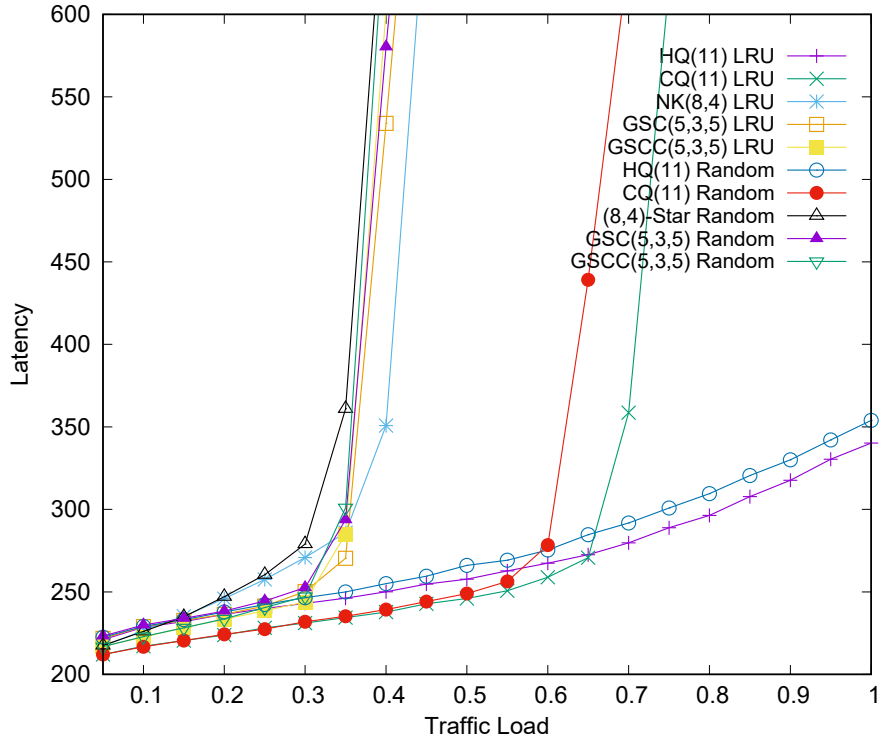

図 15: FIFO 容量が 2 のときの Random と LRU の比較

Hypercube では λ が 0.3 以上から実行時間に少しずつランダム側の実行時間が大きくなり，λ が 0.65 以上になると LRU との差が 10 となりそれ以上は差がつかず一定の変化を推移している．Crossed Cube は Hypercube とは異なり λ が 0.5 から差が大きくなる．また，λ が 0.6 のとき LRU の実行時間は 258，ランダム側では 278 であり，0.65 のときの LRU とランダム選択はそれぞれ 270 と 439 で，0.7 のときの LRU とランダム選択はそれぞれ 358 と 634 である．つまり λ が 0.7 のときの 2 つの差が 284 となり，λ が 0.65 でランダム選択のものよりも小さくなる．(n, k)-Star Graph は λ が 0.35 のときの LRU とランダム選択ではそれぞれ 285 と 361 であり，0.40 のときの LRU とランダム選択はそれぞれ 350 と 694 である．これは全トポロジの中でも最も差が大きくなり，(n, k)-Star Graph では特に衝突した際の選択方法を変えることで低次数である衝突する回数を減らすことができている．Generalized-Star Cube と Generalized-Star Cube のトポロジも Crossed Cube と (n, k)-Star Graph と同様に LRU 側が実行時間が小さくなる．しかしながら，(n, k)-Star Graph とは異なり LRU との差はあまり見られない．GSC は λ が 0.35 のときの LRU とランダム選択ではそれぞれ 270 と 294 であり，0.40 のときの LRU・ランダム選択はそれぞれ 534・580 である．また，0.35 のときの LRU とランダム選択ではそれぞれ 285 と 301 であり，0.40 のときの LRU ランダム選択はそれぞれ 601 と 669 である．

これらのことから，トポロジの種類やアルゴリズムの種類，FIFO 容量の変化だけではなく，パケット衝突した際の選択方法を変えることにより，特にトラフィック負荷が大きいときに実行時間に大きく差が出ることが判明した．更に実行時間を小さくするためには，何度もパケット同士が衝突しあい送信できない状態のときは一度すぐに送信できる異なる隣

接ノードに送信して衝突を解消できるとさらなる実行時間の減少になると考えられる．

4.6. 宛先ノードの変更による結果と考察

この小節では宛先ノードのアドレスと送信元ノードのアドレスを Hypercube と Crossed Cube 部分ではビット反転したもの，(n,k)-Star Graph 部分では先頭を同じ数字でかつ他の部分は送信元ノードにはあるが宛先ノードには存在しないアドレスに設定した．これは，パケット送信する際送信元ノードから宛先ノードに到達するまでに経由するノード数を増やす目的がある．表 3 はパケット λ が 0.05 から 0.05 刻みで 0.25 までの実行時間を示したものである．これ以上のトラフィック負荷は実行時間が更に大きくなるため検証していない．

表 3: 宛先ノードの変更による実験結果

λ	HQ(11)	CQ(11)	(8,4)-Star	GSC(5,3,5)	GSCC(5,3,5)
0.05	211	562	1065	470	521
0.10	211	777	2221	691	689
0.15	211	1015	3589	800	836
0.20	211	1256	6157	957	976
0.25	211	1475	10127	1186	1134

Hypercube は λ にかかわらず常に一定となる．これは Hypercube の探索アルゴリズムが下位ビットから変換するため，トラフィック負荷がどのような状態でもパケットごとの隣接ノードへの送信先が全て異なり衝突が発生しないためである．ゆえに実行時間は到着カウンタ数 (200) + 次数 $(11) = 211$ となる．他 4 つのトポロジを見ると，λ の増加により実行時間も増加している．宛先ノードを任意のものと比較すると Crossed Cube では λ が 0.05 のとき 350，0.10 のときは 561，0.15 のとき 795，0.20 のとき 1023，0.25 では 1259 の差となり，λ が 0.05 大きくなるたびに 250 もの実行回数が増えることが判明した．また，(n,k)-Star Graph を比較したところ任意のアドレスを指定したときよりも実行時間が大幅に増加している．これは，(n,k)-Star Graph は直径は小さいが低次数でもあることで，同じ隣接ノードにパケットを送信する際に衝突回数が発生する回数が増えてしまったためであると考えられる．GSC と GSCC で比較するとどちらも Crossed Cube よりも小さくなる．しかしながら，実行時間は λ が 0.10 と 0.25 では GSCC が GSC より小さくなる．理由としては (n,k)-Star Graph 部分は実行時間は他トポロジと比較すると非常に大きいためその部分の探索に長時間かかる．ゆえに (n,k)-Star Graph 部分の探索時間に強く依存するためであると考えられる．以上のことから，最長距離にパケットを送信するときには (n,k)-Star Graph を用いることは不向きで，利用する際も k の値を n と比べて小さくしない限り低次数で直径が大きくなり実行時間も非常に大きくなると判断できる．また，Hypercube を除いたトポロジも実行時間が大きく伸びるためあまり向いていないと考えられる．

5. 耐故障経路探索アルゴリズム

この節では，耐故障経路探索アルゴリズムとして新たに提案した 3 つの手法と実験内容，結果と考察について述べる．5.1 節では，積グラフの性質を利用した Reversible Algorithm，5.2 節では簡単な Crossed Cube と (n,k)-Star Graph の耐故障経路探索アルゴリズム，5.3 節では上記 2 つのアルゴリズムを合わせたアルゴリズムについて紹介する．

5.1. Reversible Algorithm

Reversible Algorithm とは Crossed Cube 部分の最短経路探索時に次探索のノードまたはリンク部が故障していた場合，一度 (n,k)-Star Graph 部分を探索するアルゴリズムのことである．(n,k)-Star Graph 部分を一度探索した後，再び Crossed Cube 部分を探索して宛先ノードへ近づける．Crossed Cube 部分の探索と (n,k)-Star Graph 部分の次の経路が両方とも故障していた場合，または Crossed Cube 部分が探索済みで (n,k)-Star Graph の次の経路が故障していた場合探索失敗とする．このアルゴリズムの利点として，最短経路と同じホップ数で宛先ノードに到達することができる点と 2 つの最短経路探索アルゴリズムをそのまま利用できる点が挙げられる．また，この経路探索アルゴリズムの実行時間も最短経路探索アルゴリズムと同様に O(n+k) で表すことができる．このアルゴリズムは次の Algorithm 5 に示す．Algorithm 4 との違いは Crossed Cube の探索をして次探索先の候補を見つけてから一度探索させる．そのノードとリンク部分が故障していたとき，(n,k)-Star Graph 部分の探索をする．ここでステート変数 $state$ を用意し，Crossed Cube 部分の探索で次ノードが故障していないとき，(n,k)-Star Graph 部分を探索しないようにしている．これ以外のアルゴリズムは全て最短経路の Algorithm 1 から 3 と同じである．

5.2. Pair-Related and Put-Head Algorithm

Pair-Related and Put-Head Algorithm (PRPH) は，Crossed Cube と (n,k)-Star Graph の両方に対して耐故障経路を設定したものである．Crossed Cube では最短経路探索時に故障が発生した際に反転したいビットより下のビットを確認する．ペア関係 $(R' = (10,00),(00,10))$ のどちらかが成り立つ場合，上位ビットからこれを探索するようにする．これにより経路探索数をできるだけ小さく探索することができるため優先して探索させる．ペア関係がない場合，アドレスの上位ビットから異なる部分を探索させる．これによる Crossed Cube 部分の探索が最長でも m 回となり，経由回数を抑えることができかつ到達率を向上させることができる．

また，(n,k)-Star Graph 部分の探索では宛先ノードにあり送信元ノードにないアドレスを小さい順に 1 つずつ代入している．それらの隣接ノードが全て故障により経路探索できない場合，失敗となる．このアルゴリズムの利点として，最短経路アルゴリズムと同様に Crossed Cube 部分と (n,k)-Star Graph 部分の 2 つのアルゴリズムを分けて設計することができる点が挙げられる．このアルゴリズムの実行時間は 1 回経由するために m 回または

Algorithm 5 Reversible Algorithm

Input: CQ(m), (n,k)-Star of Source node s, u
Input: CQ(m), (n,k)-Star of Destination node t, v
Output: Routing order of S
  while $s \neq t$ or $u \neq v$ **do**
    $state \leftarrow 0$
    if $s \neq t$ **then**
      $p \leftarrow$ CQ_Part(s,t)
      $state \leftarrow 1$
    end if
    if ($u \neq v$ and $state = 0$) or (p is fault node and link) **then**
      $p \leftarrow$ nkStar_Part(u,v)
    end if
    if p is not fault link and node **then**
      $S \leftarrow S + \{p\}$
    else
      return -1 ▷ Failure
    end if
  end while
  return S

$k/2$ 回かかるため，最短経路より大きくなり $O(m^2 + k^2)$ となる．また，Crossed Cube が持つパラメータ m が (n,k)-Star Graph が持つパラメータ k に対して十分大きい場合，実行時間は $O(m^2)$ となり，反対に十分小さくなる場合 $O(k^2)$ となる．これらのアルゴリズムは Algorithm 6,7,8 に示す．Algorithm 8 を実行することにより他 2 つのアルゴリズムを呼び出すことができる．

Algorithm 6 dp_Pair(s,t)

Input: CQ(m) of Source node s
Input: CQ(m) of Destination node t
Output: Address number of s
  for $i = \lfloor (m-1)/2 \rfloor - 1$ **downto** 0 **do**
    if $(s_{2i+1}s_{2i}, t_{2t+1}t2t) \in \{(10,00),(00,10)\}$ **then**
      route to s' the $2j$th of s
      if s' is not fault node or link **then return** s'
      end if
    end if
  end for
  for $i = m-1$ **downto** 0 **do**
    if $s_i \neq t_i$ **then**
      route to s' the ith of s
      if s' is not fault node or link **then**
        return s'
      end if
    end if
  end for
  return -1 ▷ Failure

Algorithm 7 Put_Head(u, v)

Input: (n, k)-Star of Source node u
Input: (n, k)-Star of Destination node v
  $P \leftarrow v \backslash u$
  find minimum P_j
  $u_0 \leftarrow P_j$
  return u

Algorithm 8 PRPH Algorithm

Input: CQ(m), (n, k)-Star of Source node s, u
Input: CQ(m), (n, k)-Star of Destination node t, v
Output: Routing order of S
  while $s \neq t$ or $u \neq v$ **do**
    if $s \neq t$ **then**
      $p \leftarrow$ CQ_Part(s, t)
      $state \leftarrow 1$
      if p is fault node and link **then**
        $p \leftarrow pair\_related(s, t)$
      end if
    else if $u \neq v$ **then**
      $p \leftarrow$ nkStar_Part(u, v)
      if p is fault node and link **then**
        $p \leftarrow put\_Head(u, v)$
      end if
    end if
    if p is not fault link and node **then**
      $S \leftarrow S + \{p\}$
    else return -1 ▷ Failure
    end if
  end while
  return S

5.3. Mixture Algorithm

Mixture Algorithm とは 5.1 節，5.2 節で紹介した耐故障経路探索アルゴリズム 2 つを合わせたものである．経路探索の順番として，はじめに Crossed Cube の最短経路を探索し，その経路部分が故障していた場合耐 Crossed Cube 用アルゴリズムを実行する．Crossed Cube 部分の探索ができない場合，(n, k)-Star Graph 部分の最短経路アルゴリズム，耐 (n, k)-Star Graph 用のアルゴリズムの順番に実行する．これら全てが探索できない場合，経路探索失敗となる．このアルゴリズムの利点として，4 つのアルゴリズムのいずれかで経路探索成功すれば経路探索し続けることができるため耐故障性に優れている点，このアルゴリズムの実行時間は 1 回経由するために $m + k/2$ 回経由するため，PRPH アルゴリズムよりも少し大きい $O((m + k)^2)$ となる．また，PRPH と同様に Crossed Cube が持つパラメータ m が (n, k)-Star Graph が持つパラメータ k に対して十分大きい場合，実行時間は $O(m^2)$ となり，反対に十分小さくなる場合 $O(k^2)$ となる．これらは，PRPH の実行時間と全く同じになる．Algorithm は次の 8 に示す．これは Crossed Cube 部分と (n, k)-Star Graph 部分の探索

時に故障ノードとリンクであった場合に耐故障性のあるアルゴリズムを実行するものになる．これらのアルゴリズムは Algorithm 6,7 と同じものを利用する．Algorithm 8 との違いとしては Crossed Cube 部分を探索したか確認する変数を 1 つ用意して探索済みではない場合も (n,k)-Star Graph 部分のアルゴリズムを実行できる点が異なる．例として，Crossed Cube 部分が故障していた場合，dp_pair，(n,k)-Star Graph の耐故障探索アルゴリズム，put_Head の順番でそれぞれ次のノードとリンクが故障しているかを確認するアルゴリズムになる．

Algorithm 9 Mixture Algorithm

Input: CQ(m), (n,k)-Star of Source node s，u
Input: CQ(m), (n,k)-Star of Destination node t，v
Output: Routing order of S
  while $s \neq t$ or $u \neq v$ **do**
    $state \leftarrow 0$
    if $s \neq t$ **then**
      $p \leftarrow$ CQ_Part(s,t)
      $state \leftarrow 1$
      if p is fault node and link **then**
        $p \leftarrow pair\_related(s,t)$
      end if
    end if
    if ($u \neq v$ and $state = 0$) or (p is fault node and link) **then**
      $p \leftarrow$ nkStar_Part(u,v)
      if p is fault node and link **then**
        $p \leftarrow put\_Head(u,v)$
      end if
    end if
    if p is not fault link and node **then**
      $S \leftarrow S + \{p\}$
    else **return** -1 ▷ Failure
    end if
  end while
  return S

5.4. 最短経路探索アルゴリズムの実験内容

実際のスーパーコンピュータ内のパソコンまたは接続部分が故障してその部分が利用できないことを想定し，トポロジのノードまたはリンクの一部が故障していて経路探索するときその部分を経由することを不可能とする．そのとき，別の経路を探索できない場合，探索失敗と判定する．ここではノード故障の場合は送信元ノードと宛先ノードは故障しないものとする．また，リンク故障は送信元ノードと宛先ノードが孤立している場合も考える．ここで、ノード・リンクの故障率を設定し、5%刻みで 5%から 95%まで増加させる．これらを最短経路探索アルゴリズムと今回提案した 3 つのアルゴリズムとで 10000 回繰り返し，これらを比較して到達率の変化を調べる．また，到達成功時の平均ホップ数を調べ，最短経路探索アルゴリズムとの差を調べる．今回使用する Generalized-Star Crossed Cube のパラ

メータ n, k, m をそれぞれ $5, 3, 7$ と設定した．それによるノード数，次数，直径はそれぞれ 7680，12，9 となる．

5.5. ノード故障の場合の結果と考察

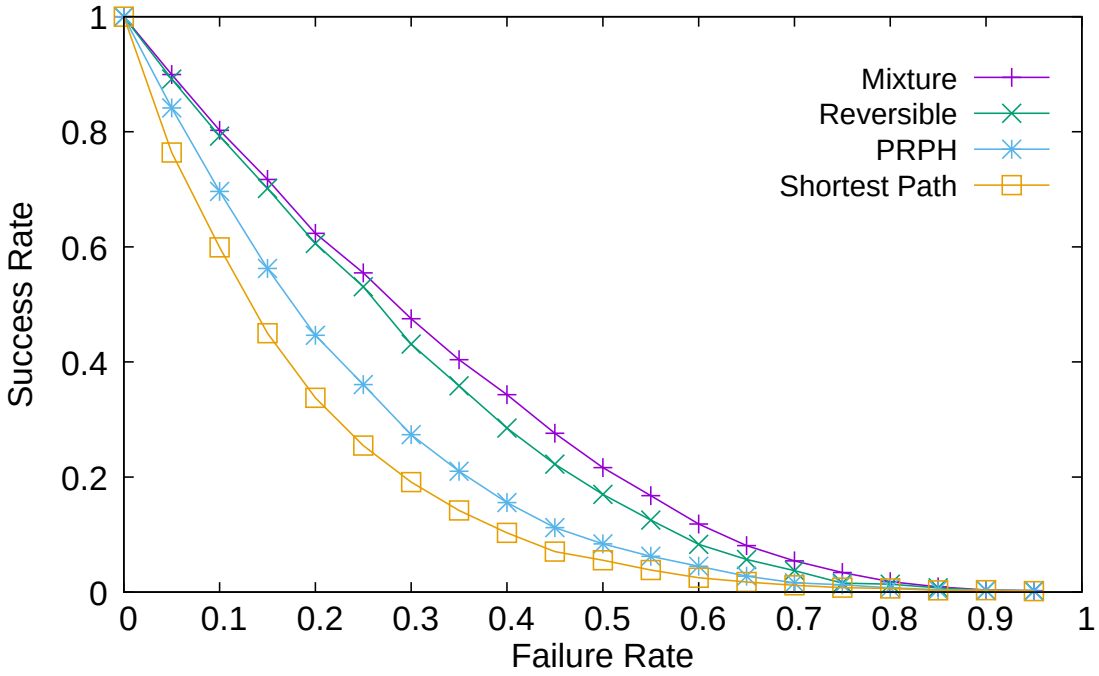

図 16: ノード故障のときの到達成功率

図 16 と図 17 はそれぞれノード故障のときの故障率による到達率と平均ホップ数を示したものである．ノード故障の場合，最短経路探索アルゴリズムでは故障率が増加すると同時に急激に到達率が減少している．PRPH は最短経路と同様に故障率が増加すると急激に減少しているが故障率 20%のとき到達率がおよそ 10%の差を生じている．Reversible は故障率が 50%まではほぼ直線的に到達率が減少していて特に故障率が 30%のとき最短経路との差が最も大きくなる．Mixture は Reversive と比較すると故障率が 40%のときが最も差が大きくなる．最短経路と比較すると，故障率が 25%で最短経路の到達率は 25%であるが Mixture の到達率は 55%と 30%到達率が向上していることが判明した．

また，図 17 では平均ホップ数のグラフを示している。故障率 0%からの平均ホップ数はおよそ 7 でありホップ数が 1 小さくなるまでに最短経路は故障率 40%であるが，PRPH は故障率 50%，Reversible は故障率 60%，Mixture は故障率 70%で 1 小さくなっていることが確認できる．故障率 75%のとき最短経路と Mixture の平均ホップ数の差が最も大きくなり，その差は 2.7 となる．このことから送信元ノードから宛先ノードまでの距離が大きくても到達できる回数が増え，少し遠回りしても到達することができると判断できる。

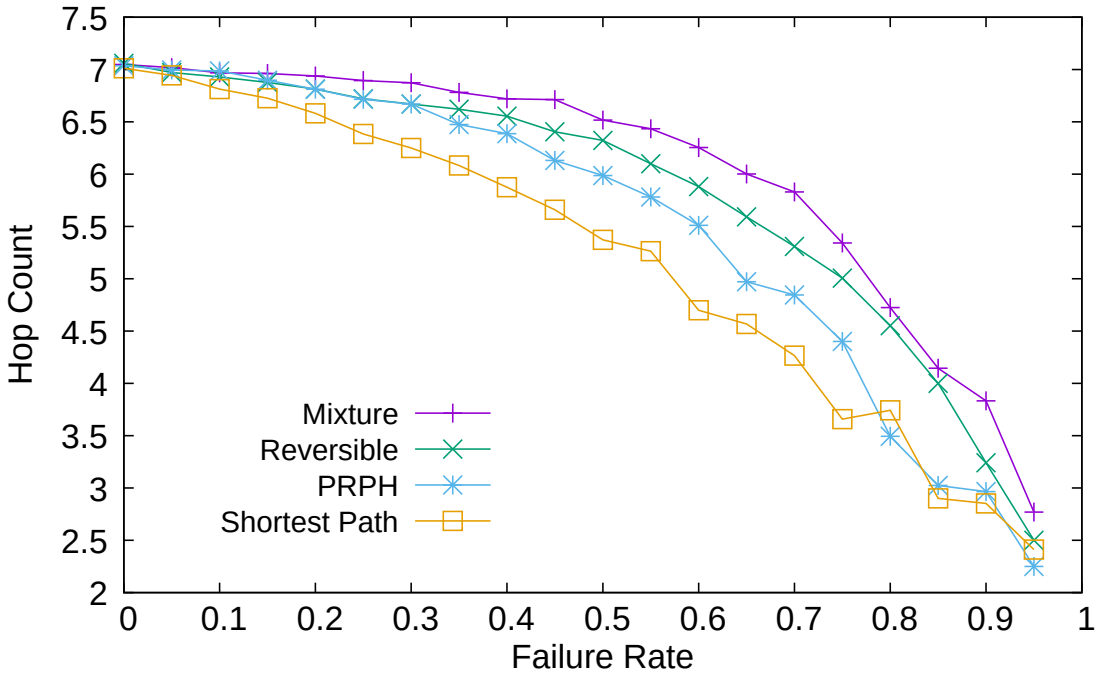

図 17: ノード故障のときのの平均ホップ数

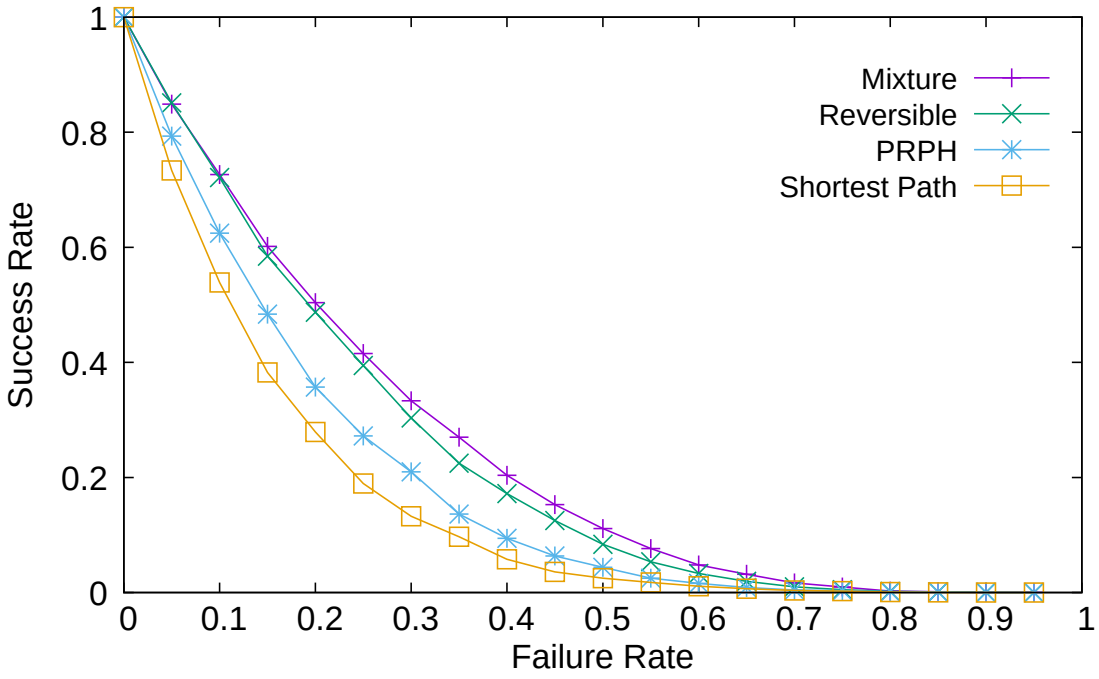

図 18: リンク故障の到達成功率

5.6. リンク故障の場合の結果と考察

図 18 と図 19 はそれぞれリンク故障のときの故障率による到達率と平均ホップ数を示したものである．ノード故障と比較すると全体的に到達率が低下している．これは実験節に述べた送信元ノードまたは宛先ノードの全てのリンクが故障していた状態のとき，到達不

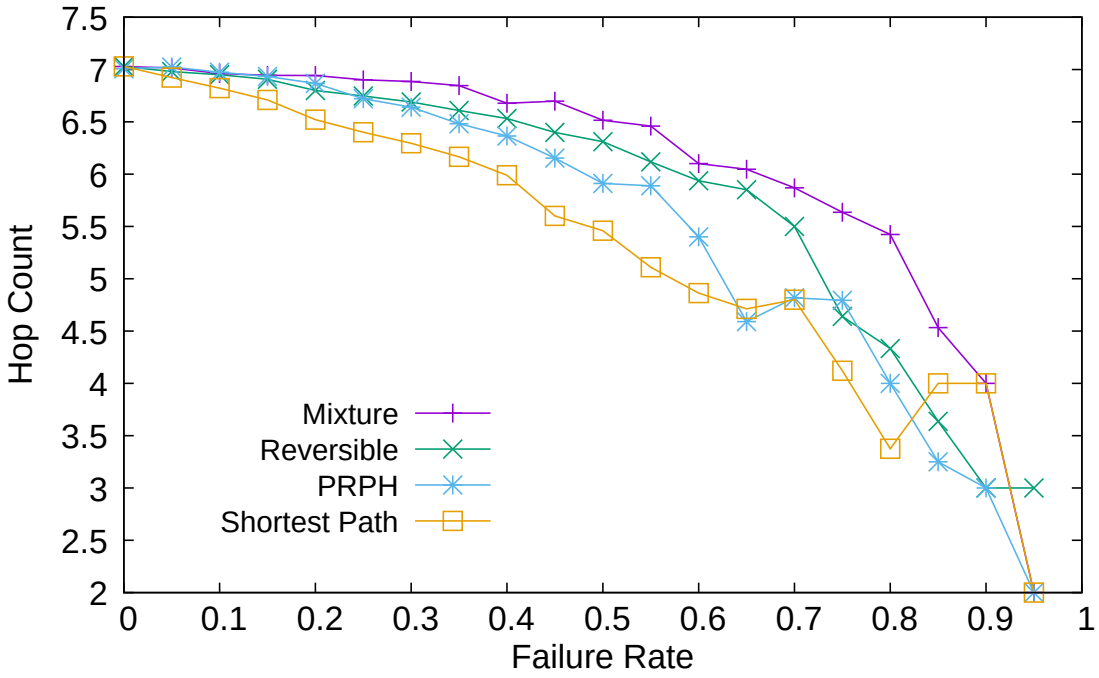

図 19: リンク故障のときの平均ホップ数

能になるためであると考えられる．最短経路探索アルゴリズムでは，故障率が 20%のとき到達率は 28%であるが，PRPH は 36%で 8%到達率が向上し，Reversible と Mixture では到達率が 50%と 20%到達率が向上していることが確認できる．Reversible と Mixture を比較すると故障率 35%のとき最も差が大きくなり，その差は 5%である．

また，図 19 の平均ホップ数では故障率 0%のときの平均ホップ数はおよそ 7 であり，平均ホップ数が 1 減少するのに最短経路では故障率 40%，PRPH は故障率 50%，Reversible は故障率 60%，Mixture は故障率 70%である．これはノード故障のときと同じである．しかしながら，故障率 60%を超えたあたりから Mixture を除いた 3 つのアルゴリズムの平均ホップ数が大きく異なっている．これは図 19 から最短経路の到達率が 10000 回中 100 回未満と非常に小さい結果を示していたため母数が少ないために生じたためである考えられる．それに対して Mixture は図 17 と同様にどの故障率でも安定した平均ホップ数を記録している．また他のアルゴリズムと比較しても最大で 1 以上の差を付けていることから，到達率が優れていると考えられる．

6. まとめと今後の課題

本論文では Average Packet Latency による Generalized-Star Crossed Cube の特性，および本トポロジの耐故障経路探索アルゴリズムによる到達率を向上させる 3 つのアルゴリズムを提案した．Average Packet Latency を実行したところ，トラフィック負荷が小さい場合，直径が小さいことで経由するノードが少なくなるため他トポロジよりも実行時間に有用性があることを示した．また，耐故障経路探索アルゴリズムでは Crossed Cube 部分と (n, k)-Star Graph 両方の耐故障経路探索アルゴリズムを提案し，さらに積グラフの特性を活かし 2 つの耐故障経路探索アルゴリズムを混ぜて探索することで特にノード故障のときの到達率を最大で 30%向上することができた．しかしながら，パケット送信する際にトラフィック負荷が大きくなると，パケット同士の衝突によるデッドロックが頻繁に発生して処理に時間がかかることやパケット送信ができなくなるなどの問題点が挙げられる．それを回避するためのデッドロックを考慮したものあるいはデッドロックフリーのアルゴリズムを提案することが重要になる．また，耐故障経路の Crossed Cube 部分と (n, k)-Star Graph のアルゴリズムはどちらも数ステップ戻って探索し直す操作を実装していないため，さらに到達率を向上させるためにはこのような操作も重要である．今後の課題として，これら 2 つを満たして更に性能向上できるアルゴリズムを提案することが必要になる．

参考文献

[1] T. Sato and Y. Li, “Generalized-star crossed cube - a flexible interconnection network with high-performance at low-cost,” in *2017 Fifth International Symposium on Computing and Networking (CANDAR)*, Nov 2017, pp. 153–158.

[2] Y. Li and W. Chu, “Mikant: A mirrored k-ary n-tree for reducing hardware cost and packet latency of fat-tree and clos networks,” in *IEEE ScalCom 2018*, 10 2018, pp. 1643–1650.

[3] Y. Saad and M. H. Schultz, “Topological properties of hypercubes,” *IEEE Transactions on Computers*, vol. 37, no. 7, pp. 867–872, Jul 1988.

[4] K. Efe, “The crossed cube architecture for parallel computation,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 3, no. 5, pp. 513–524, Sep. 1992. [Online]. Available: http://dx.doi.org/10.1109/71.159036

[5] W.-K. CHIANG and R.-J. CHEN, “Topological properties of the (n,k)-star graph,” *International Journal of Foundations of Computer Science*, vol. 09, no. 02, pp. 235–248, 1998. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S0129054198000167

[6] D. Arai and Y. Li, “Generalized-star cube: A new class of interconnection topology for massively parallel systems,” in *2015 Third International Symposium on Computing and Networking (CANDAR)*, Dec 2015, pp. 68–74.

[7] N. Adhikari and C. Ranjan Tripathy, “Star-crossed cube: An alternative to star graph,” *TURKISH JOURNAL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCES*, vol. 22, pp. 719–734, 01 2014.

[8] C.-P. Chang, T.-Y. Sung, and L.-H. Hsu, “Edge congestion and topological properties of crossed cubes,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 1, pp. 64–80, Jan 2000.

付録

A.1. 使用したコードについて

使用したコードを付録に紹介すると非常に長くなるため，今回使用したコードは全て https://github.com/Tomdemort/thesis_code.git に入っています．下記は Average Packet Latency のメインメソッドになります.

A.2. controller5.java

```
1  package controller5;
2
3  import java.math.*;
4  import java.util.*;
5
6  public class Controller5 {
7
8    Random rand = new Random(5L);
9    static double lambda = 0.05;
10   static int nkType = 0;
11
12   public static void main(String[] args) {
13     int conflictCount = 0;
14     for(int type = 1;type <= 5; type++) {
15       int[] nkm = new int[3];
16       if(type == 5 || type == 4) {
17         nkm[0] = 5; nkm[1] = 3; nkm[2] = 5;
18       }
19       else if(type == 3) {
20         nkm[0] = 8; nkm[1] = 4; nkm[2] = 1;
21       }
22       else if(type == 1 || type == 2) {
23         nkm[0] = 1; nkm[1] = 1; nkm[2] = 11;
24       }
25       else if(type == 6) {
26         nkm[0] = 5; nkm[1] = 1; nkm[2] = 1;
27       }
28       int N = Topology.calcN(type, nkm);
29       System.out.println("#Type: " + type+",\t Latency,\t Conflict");
30       ArrayList<Node> nodes = new ArrayList<Node>();
31       while(lambda <= 1.0) {
32         int count = 0;
33         int eCount = 0; //errorCount;
34         conflictCount = 0;
35         for(int i = 0;i<100;i++) {
36           nodes  = new ArrayList<Node>();
37           for (int j = 0;j<N;j++) {
38             Node node = new Node(type,j,nkm);
39             node.setNKType(nkType);
40             nodes.add(node);
41           }
```

```
42            Topology topo = new Topology(nodes,type,nkm);
43            topo.setNKType(nkType);
44            int[] runList = new Controller5().makeRunList(N);
45            topo.init(runList);
46            int tmp = topo.run();
47            if(tmp == -1) {
48              eCount++;
49              i--;
50            }
51            conflictCount += topo.getAllConflictCount();
52            count += tmp;
53          }
54          System.out.print(lambda +", \t " + (count * 1.0) /(100 - eCount));
55          System.out.println(", \t " + (conflictCount * 1.0) /(100 - eCount));
56          if(eCount >= 1) System.out.println("lambda=" + lambda +": ErrorCount:" +
                eCount);
57          BigDecimal a = new BigDecimal(lambda);
58          BigDecimal b = new BigDecimal("0.05");
59          b = a.add(b).setScale(2,RoundingMode.HALF_UP);
60          lambda = b.doubleValue();
61          }
62        lambda = 0.05;
63        }
64    }
```