ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 5, Issue 1, pp: (25-32), Month: January - March 2017, Available at: www.researchpublish.com

Computation of Maximum Flow Using Preflow Push Algorithm in PIM-DM Wireless Multicast

[1]Deepa V B, [2]Usha Devi M B

Department of Information Science and Department of Telecommunication,
Jawaharlal Nehru National College of Engineering Shimogga

Abstract: **PIM-DM is a multicast routing protocol that uses the underlying unicast routing information base to flood multicast datagrams to all multicast routers. Prune messages are used to prevent future messages from propagating to routers without group membership information. There are several polynomial-time algorithms for the maximum flow problem. One inherent drawback of these augmenting path algorithms is the computationally expensive operation of sending flows along paths. These algorithms might repeatedly augment flows along common path segments. The pre flow-push algorithms that we described in this paper overcome this drawback; we can conceive of them as sending flows along several paths simultaneously in PIM-DM wireless multicast.**

Keywords: **Mulicast, Maximum Flow, Pre Flow, PIM-DM.**

1. INTRODUCTION

Traditional IP communications categorized into two modes: point-to-point communications between a source host and a destination host, known as unicast, and point-to-multipoint communications between a source host and all other hosts on the same subnet with the source host, known as broadcast. With broadcast, the information is delivered to all hosts, rather than some specific hosts that need the information, resulting in waste of network bandwidth. In addition, broadcasts are confined only to the local subnet. With unicast, as a separate copy of information is sent to each host, the duplicate IP packets not only use a tremendous amount of network bandwidth but also add to the burden of the source host. Therefore, the conventional unicast and broadcast technologies cannot effectively address the issue of point-to-multipoint data transmission.

Multicast provides a best-effort service to deliver data packets to a specific set of receiver hosts, known as a multicast group, on the network. With multicast, the source host, known as a multicast source, sends only one copy of data packets destined for a multicast group address, and each receiver host of the multicast group can receive the data packets. Only the hosts that have joined the multicast group can receive the traffic addressed to the multicast group, while hosts out of the multicast group cannot.

1.1 Benefits:

Compared with unicast and broadcast, the multicast technique effectively addresses the issue of point-to-multipoint data transmission. By allowing high-efficiency point-to-multipoint data transmission over an IP network, multicast greatly saves network bandwidth and reduces network load. More importantly, multicast allows convenient deployments of new value-added services in Internet-based information service areas, such as live Webcasting, Web TV, distance learning, telemedicine, Web radio, and real-time videoconferencing.

1.2 Multicast Routing Protocols:

Similar to unicast protocols, multicast routing protocols fall into intra-domain and inter-domain protocols:

- Based on the group memberships maintained by IGMP, an intra-domain multicast routing protocol builds multicast distribution trees according to certain multicast routing algorithms and creates multicast routing state entries on multicast routers, which forward multicast traffic as per the routing state entries.
- Based on the inter-domain multicast routing policy configured in the network, an inter-domain multicast routing protocol propagates multicast source information and exchanges multicast routing information among autonomous systems (ASs), thus ensuing multicast forwarding among different domains.

1.3 Intra-Domain Multicast Routing Protocols:

Among a variety of intra-domain multicast routing protocols, Protocol Independent Multicast (PIM) is a popular one. Based on the forwarding mechanism, PIM falls into two modes – dense mode (referred to as PIM-DM) and sparse mode (referred to as PIM-SM).

2. PIM-DM

In a PIM-DM[15] domain, routers use periodical PIM Hello messages for PIM neighbor discovery, determination of leaf routers and leaf networks, and designated router (DR) election on a multi-access network. Although PIM-DM does not require a DR, one must be elected among multiple routers on a multi-access network running IGMPv1 in a PIM-DM domain to act as the IGMPv1 querier on that multi-access network. As a dense mode multicast routing protocol, PIM-DM uses the "push" mode for multicast forwarding, and is suitable for small-sized networks with densely distributed multicast members. PIM-DM works as follows as shown in figure 1.

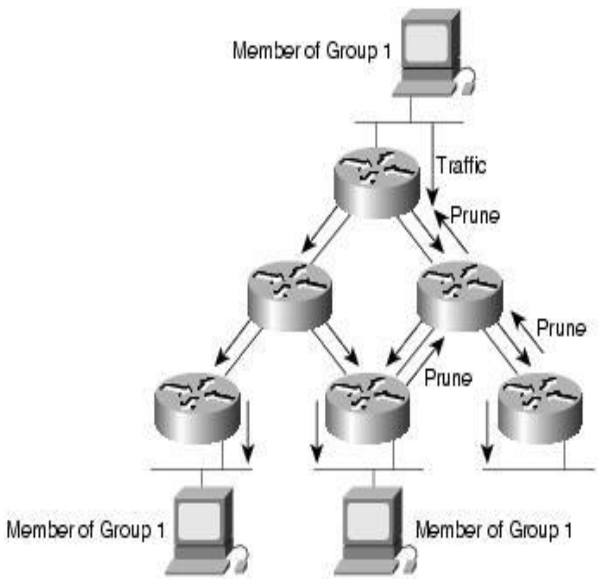

Figure 1:Protocol Independent Mode in Dense Mode

- PIM-DM assumes that at least one multicast group member exists on each subnet of a network, and therefore multicast data is flooded to all nodes on the network. Then, branches without receivers downstream are pruned from the forwarding tree, leaving only those branches with receivers. This "flood and prune" process takes place periodically, that is, pruned branches resume multicast forwarding when the pruned state times out and then data is re-flooded down these branches, and then are pruned again.
- When a host attached to a pruned node joins the multicast group, the node sends a graft message toward the upstream node. Then the node resumes multicast traffic forwarding.

2. 1 PIM-SM:

In a PIM-SM[15] domain, routers periodically sends PIM Hello messages for PIM neighbor discovery and DR election on a multi-access network, where the DR sends join/prune messages toward the root of the multicast forwarding tree for the receiver host attached to it, or forwards multicast traffic from the directly connected multicast source onto the multicast distribution tree.As a sparse mode multicast routing protocol, PIM-SM uses the "pull" mode for multicast forwarding, and is suitable for large- and medium-sized networks with sparsely and widely distributed multicast members. The basic implementation of PIM-SM is as follows as shown in figure 2.

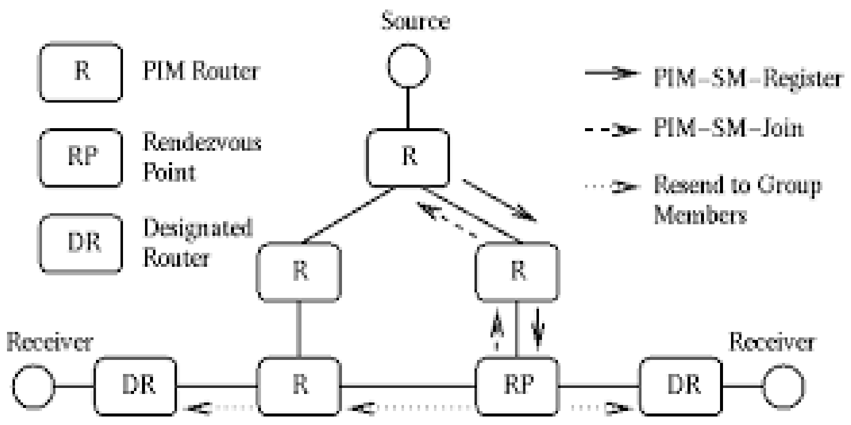

Figure 2: Protocol Independent Mode in Sparse Mode

- ❖ PIM-SM assumes that no hosts need to receive multicast data. In the PIM-SM mode, and delivers multicast data only to those hosts that have explicitly requested for the data. The core task for PIM-SM in multicast forwarding is to build and maintain rendezvous point trees (RPTs). An RPT is rooted at a router in the PIM domain as the common node, referred to as the rendezvous point (RP), through which the multicast data travels down the RPT to the receivers.
- ❖ When a receiver is interested in the multicast data addressed to a specific multicast group, the router connected to this receiver sends a join message to the RP for that multicast group as shown in figure 2. The path along which the message goes hop by hop to the RP forms a branch of the RPT.
- ❖ When a multicast source sends multicast data to a multicast group, the source-side DR first registers the multicast source with the RP by sending register messages to the RP by unicast. The arrival of a register message at the RP triggers the establishment of an SPT. Then, the multicast source sends subsequent multicast packets along the SPT to the RP. Upon reaching the RP, the multicast packet is duplicated and delivered to the receivers down the RPT.

2.2 Inter-Domain Multicast Routing Protocols:

Inter-domain multicast routing protocols are used to propagate multicast information among different ASs. So far, mature solutions include:

- ❖ Multicast Border Gateway Protocol (MBGP) is used for exchanging multicast routing information between ASs.
- ❖ Multicast Source Discovery Protocol (MSDP) is used for advertising multicast source information between Internet service providers (ISPs).

3. RELATED WORK

The maximum flow problem is distinguished by the long succession of research contributions that have improved on the worst-case complexity of the best known algorithms. Indeed, no other network flow problem has witnessed as many incremental improvements. The following discussion provides a brief survey of selective improvements; Ahuja, Magnanti, and Orlin [1] give a more complete survey of the developments in this field.The labeling algorithm of Ford and Fulkerson [1956a] runs in pseudopolynomial time. Edmonds and Karp [2] suggested two polynomial-time implementations of this algorithm. The first implementation, which augments flow along paths with the maximum residual capacity, performs O(m log U) iterations. The second implementation, which augments flow along shortest paths, performs *O(nm)* iterations. and runs in $O(nm^2)$ time. Independently, Dinic [3] introduced a concept of shortest path networks (in number of arcs), called *layered networks*, and obtained an $O(n^2m)$*-time* algorithm. Until this point all maximum flow algorithms were augmenting path algorithms. Karzanov [4] introduced the first preflow-push algorithm on layered networks; he obtained an $O(n^3)$ algorithm. Shiloach and Vishkin [5] described another *O(n 3)* preflow-push algorithm for the maximum flow problem, which is a precursor of the FIFO preflow-push algorithm The capacity scaling described is due to Ahuja and Orlin [1];this algorithm is similar to the bit-scaling algorithm due to Gabow [6] . Due to Ahuja and Orlin [1]; this algorithm can be regarded as avariant of Dinic's [3] algorithm and uses distance labels instead of layered networks.Researchers obtained further improvements in the running times of the maximum flow algorithms by using distance labels instead oflayered networks. Goldberg [8] first introduced distance labels; by incorporating them in the algorithm of Shiloach and Vishkin [5], he obtained the $O(n^3)$*-time* FIFO implementation . The generic preflow-push algorithm and its highestlabel preflow-push implementation that we described in Goldberg and Tarjan [9]. Using a dynamic tree data structure developed by Sleator and Tarjan [10], Goldberg and Tarjan [9] improved the running time of the FIFO implementation to $O(nm \log(n^2/m))$. Using a clever analysis, Cheriyan and Maheshwari [11] show that the highest-label preflow-push algorithm in fad runs in $O(n^2\sqrt{m})$ time.

Worst-case bounds of algorithms investigated

S no.	Algorithm	Discoverer(s)	Running time
1	Dinic's algorithm	Dinic (1970)	*0(n 2 m)*
2	Karzanov's algorithm	Karzanov (974)	*O(n 3)*
3	Shortest augmenting path	Ahuja and Orlin (1991)	*O(n **m**)*
4	Capacity-scaling Preflow-push algorithms	Gabow (1985) and Ahuja and Orlin (I991)	*O(nm* log U)
5	Highest-label algorithm	Goldberg and Tarjan (1986)	*O(n 2ml/ 2)*
6	FIFO algorithm	Goldberg and Tarjan (1986)	*O(n 3)*
8	Original excess-scaling	Ahuja et al. (1989)	*O(nm* + n2log *U)*
9	Stack-scaling algorithm	Ahuja et al. (1989)	***0(nn*** *t* ((n2log *U)/(log* log *)))*
10	Wave-scaling algorithm	Ahuja et al. (1989)	*O(nn + n 2* lo *u*)

4. GENERIC PREFLOW PUSH ALGORITHMS

These algorithms are more general, more powerful[12], and more flexible than augmenting path algorithms. The best preflow-push algorithms currently outperform the best augmenting path algorithms in theory as well as in practice. The inherent drawback of the augmenting path algorithms is the computationally expensive operation of sending flow along a path, which requires $O(n)$ time in the worst case. Preflow-push algorithms do not suffer from this drawback and obtain dramatic improvements in the worst-case complexity. The push–relabel algorithm (alternatively, preflow–push algorithm) is an algorithm for computing maximum flows. The name "push–relabel" comes from the two basic operations used in the algorithm. Throughout its execution, the algorithm maintains a "preflow" and gradually converts it into a maximum flow by moving flow locally between neighbouring nodes using *push* operations under the guidance of an admissible network maintained by *relabel* operations. The push–relabel algorithm is considered one of the most efficient maximum flow algorithms. The generic algorithm has a strongly polynomial $O(V^2E)$ time complexity, which is asymptotically more efficient than the $O(VE^2)$ Edmonds–Karp algorithm. Specific variants of the algorithms achieve even lower time complexities.

The push–relabel algorithm has been extended to compute minimum cost flows. The idea of distance labels has led to a more efficient augmenting path algorithm, which in turn can be incorporated back into the push–relabel algorithm to create a variant with even higher empirical performance. The algorithm starts by creating a residual graph, initialising the preflow values to zero and performing a set of saturating push operations on residual arcs exiting the source,(s, v) where $v \in V \setminus \{s\}$. Similarly, the labels are initialised such that the label at the source is the number of nodes in the graph, $\ell(s) = |V|$, and all other nodes are given a label of zero. Once the initialisation is complete the algorithm repeatedly performs either the push or relabel operations against active nodes until no applicable operation can be performed.

4.1 Push:

The push operation applies on an admissible out-arc (u, v) of an active node u in G_f. It moves $\min\{x_f(u), c_f(u,v)\}$ units of flow from u to v.

```
push(u, v):
  assert $x_f$[u] > 0 and ℓ[u] == ℓ[v] + 1
  Δ = min($x_f$[u], c[u][v] - f[u][v])
  f[u][v] += Δ
  f[v][u] -= Δ
  $x_f$[u] -= Δ
  $x_f$[v] += Δ
```

A push operation that causes *f* (*u*, *v*) to reach *c*(*u*, *v*) is called a saturating push since it uses up all the available capacity of the residual arc. Otherwise, all of the excess at the node is pushed across the residual arc. This is called an unsaturating or non-saturating push.

4.2 Relabel:

The relabel operation applies on an active node *u* without any admissible out-arcs in G_f. It modifies $\ell(u)$ to be the minimum value such that an admissible out-arc is created. Note that this always increases $\ell(u)$ and never creates a steep arc, which is an arc (u, v) such that $c_f(u, v) > 0$, and $\ell(u) > \ell(v) + 1$.

```
relabel(u):
  assert $x_f$[u] > 0 and ℓ[u] <= ℓ[v] for all v such that c[u][v] - f[u][v] > 0
  ℓ[u] = min(ℓ[v] for all v such that c[u][v] - f[u][v] > 0) + 1
```

4.3 Effects of push and relabel:

After a push or relabel operation, ℓ remains a valid labeling function with respect to *f*.For a push operation on an admissible arc (u, v), it may add an arc (v, u) to E_f, where $\ell(v) = \ell(u) - 1 \le \ell(u) + 1$; it may also remove the arc (u, v) from E_f, where it effectively removes the constraint $\ell(u) \le \ell(v) + 1$.To see that a relabel operation on node *u* preserves the validity of $\ell(u)$, notice that this is trivially guaranteed by definition for the out-arcs of *u* in G_f. For the in-arcs of *u* in G_f, the increased $\ell(u)$ can only satisfy the constraints less tightly, not violate them.

The generic push–relabel algorithm is used as a proof of concept only and does not contain implementation details on how to select an active node for the push and relabel operations. This generic version of the algorithm will terminate in $O(V^2E)$.Since $\ell(s) = |V|$, $\ell(t) = 0$, and there are no paths longer than $|V| - 1$ in G_f, in order for $\ell(s)$ to satisfy the valid labelling condition *s* must be disconnected from *t*. At initialisation, the algorithm fulfils this requirement by creating a pre-flow *f* that saturates all out-arcs of *s*, after which $\ell(v) = 0$ is trivially valid for all $v \in V \setminus \{s, t\}$. After initialisation, the algorithm repeatedly executes an applicable push or relabel operation until no such operations apply, at which point the pre-flow has been converted into a maximum flow.

```
generic-push-relabel(G, c, s, t):
  create a pre-flow f that saturates all out-arcs of s
  let ℓ[s] = |V|
  let ℓ[v] = 0 for all v ∈ V \ {s}
  while there is an applicable push or relabel operation
```

Execute the operation.

4.4 Correctness:

The algorithm maintains the condition that ℓ is a valid labeling during its execution. This can be proven true by examining the effects of the push and relabel operations on the label function ℓ. The relabel operation increases the label value by the associated minimum plus one which will always satisfy the $\ell(u) \le \ell(v) + 1$ constraint. The push operation can send flow from u to v if $\ell(u) = \ell(v) + 1$. This may add (v, u) to G_f and may delete (u, v) from G_f. The addition of (v, u) to G_f will not affect the valid labeling since $\ell(v) = \ell(u) - 1$. The deletion of (u, v) from G_f removes the corresponding constraint since the valid labeling property $\ell(u) \leq \ell(v) + 1$ only applies to residual arcs in G_f.If a preflow f and a valid labeling ℓ for f exists then there is no augmenting path from s to t in the residual graph G_f. This can be proven by contradiction based on inequalities which arise in the labeling function when supposing that an augmenting path does exist. If the algorithm terminates, then all nodes in $V \setminus \{s, t\}$ are not active. This means all $v \in V \setminus \{s, t\}$ have no excess flow, and with no excess the preflow f obeys the flow conservation constraint and can be considered a normal flow. This flow is the maximum flow according to the max-flow min-cut theorem since there is no augmenting path from s to t.Therefore, the algorithm will return the maximum flow upon termination.

The following is a sample execution of the generic push-relabel algorithm, as defined above, on the following simple network flow graph diagram is shown in figure 4.a and 4.b.

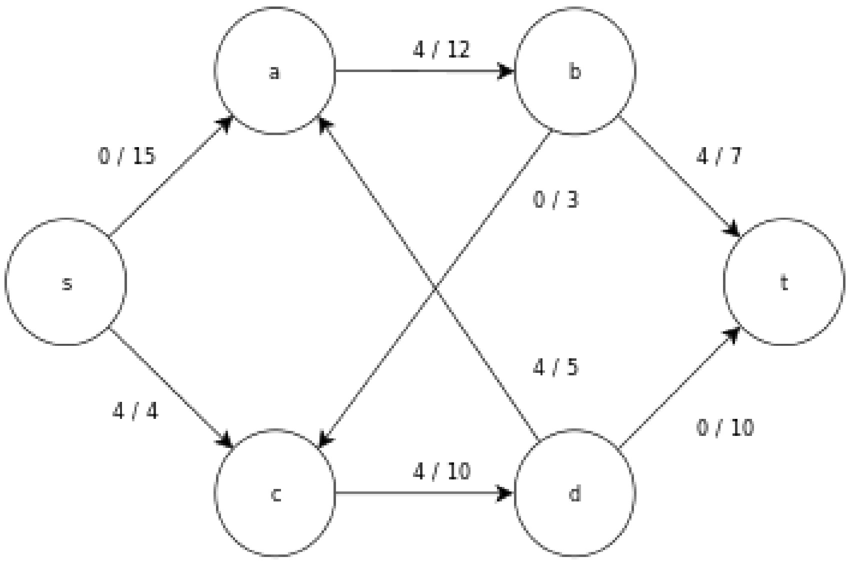

Figure 4.a Initial Flow Network Graph

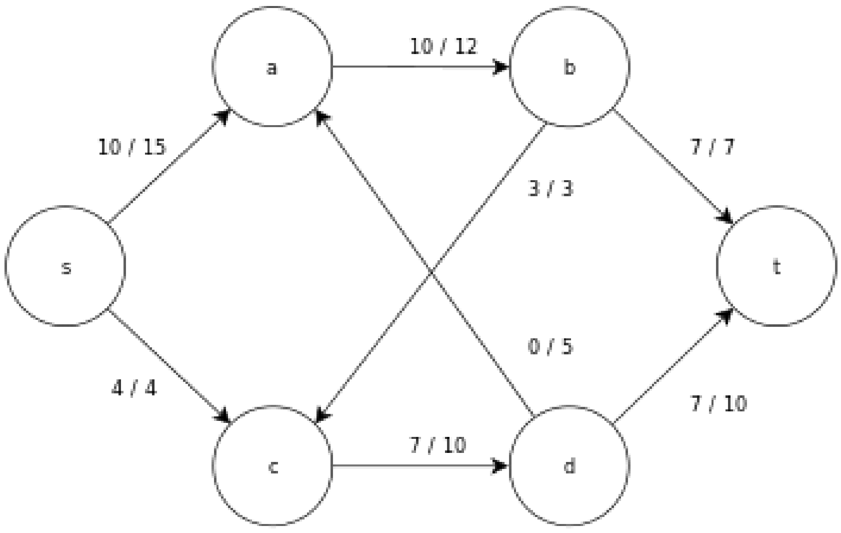

Figure 4.b Final Maximum Flow Network Graph

5. RESULT ANALYSIS

We compare the following preflow-push algorithms with capacity scaling, Dinic, shortestPath, excess-scaling algorithms. Out of these, we focus more on the preFlow algorithms, which empirically is the fastest of the algorithms as shown in Figure 5.1a.

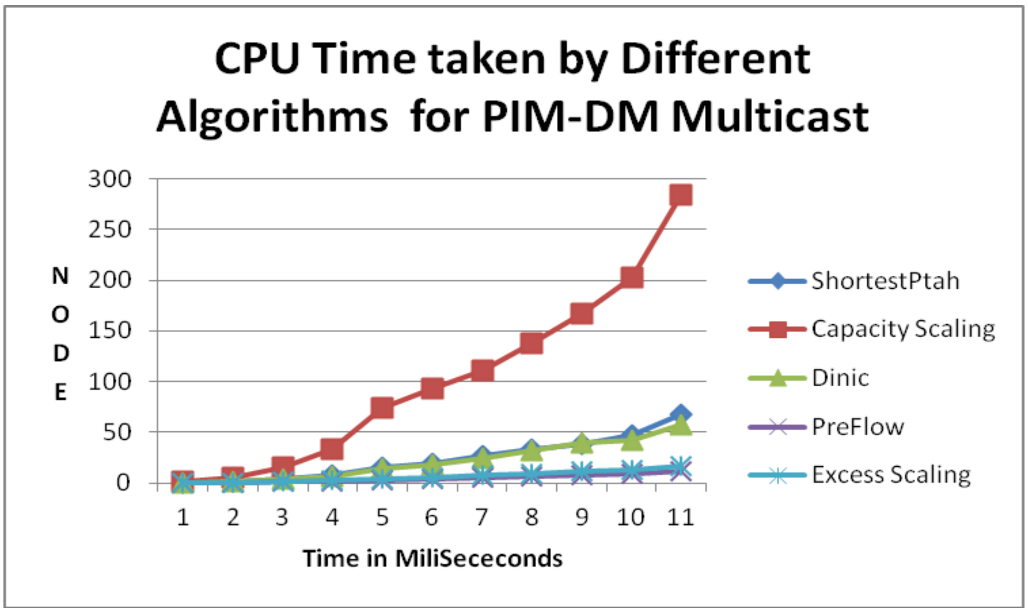

Figure 5.a CPU times taken by the Preflow-push algorithm

In Figure 5.a we show the CPU times taken by the preflow-push algorithm for PIM-DM Multicast networks.

5.1 Arc Relabel:

We have shown earlier that for the preflow-push algorithms, the representative operations are (i) α_r the arc-relabels, and (ii) α_p the pushes. To identify the asymptotic bottleneck operation, we plot the ratio $\alpha_r/(\alpha_r+\alpha_p)$ .This graph suggests that arc-relabels grow at a slightly faster pace than the pushes. This contrasts with the worst-case analysis where arc-relabels grow at a much slower pace than the pushes. This finding for the preflow-push algorithm is similar to our finding for the shortest augmenting path algorithm where we observed that arc-relabels take more time empirically than the arc-augmentations.

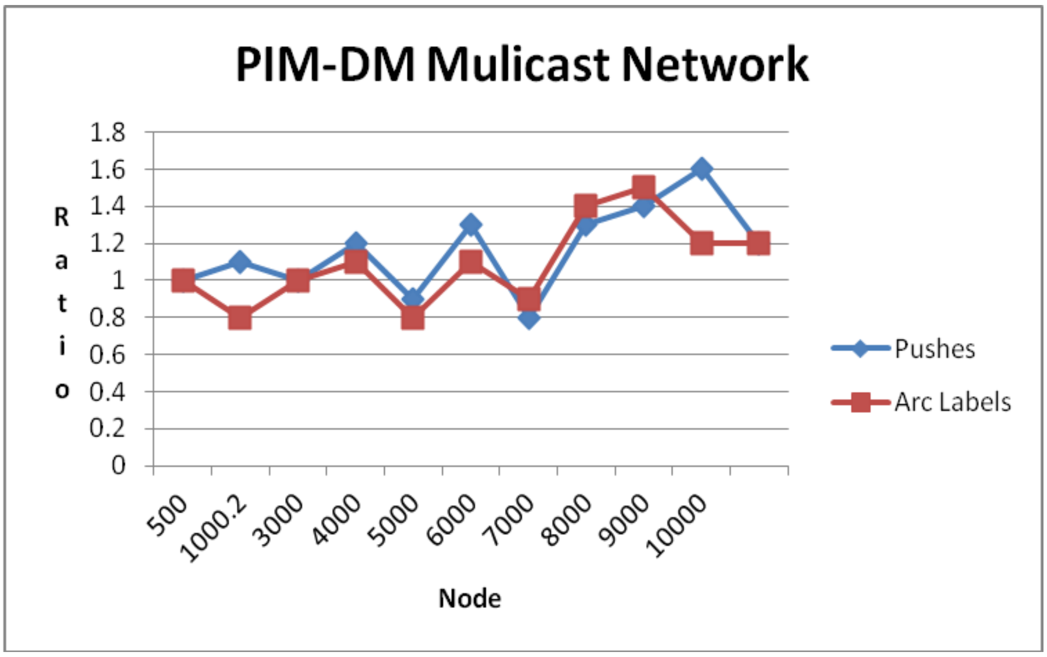

Figure 5.b Pushes and Arc Labels

Figure 5.b shows ratio of predicted to actual arc-labels and pushes for the PreFlow-push alorithms.

6. CONCLUSION

The model of PIM-DM multicast applies so called “push” model of data delivering. The active source multicast data to the whole network and the particular receivers have to announce their interest by means of the Graft message or in case the lack of interest with the Prune .The preflow-push algorithm is fastest among the algorithms tested by us, which can compute maximum flow in PIM-DM Source Specific Network efficiently

REFERENCES

[1] Ahuja, R.K., and Orlin, J.B. (1996), "Use of representative operation counts in computational testings of algorithms", *ORSA Journalof Computing 8/3*

[2] EDMONDS, J., and R. M. KARP. 1972. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal ofACM* 19,248-264

[3] DINIC, E. A. 1970. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Mathematics Doklady* 11, 1277-1280.

[4] KARZANOV, A. V. 1974. Determining the maximal flow in a network by the method of preflows. *Soviet Mathematics Doklady* 15, 434-437.

[5] SHiLOACH, Y., and U. VISHKIN. 1982. An *O(n2* log *n)* parallel max-flow algorithm. *Journal of Algorithms* , 3, 128-146.

[6] GOLDBERG, A. V., and R. E. TARJAN. 1986. A new approach to the maximum flow problem. *Proceedingsof the 18th ACM Symposium on the Theory of Computing*, pp. 136-146. Full paper in *Journal of ACM* 35(1988), 921-940.

[7] An Arc hitecture for Wide-Area Multicast Routing, S. Deering, D. Estrin, D. F arinacci, V. Jacobson, G. Liu,L. W ei, USC T ec hnical Rep ort, a v ailable from authors, F eburary 1994

[8] Host Extensions for IP Multicasting, Net w ork W orking Group, RF C 1112, S. Deering, August 1989

[9] GOLDBERG, A. V., and R. E. TARJAN. 1987. Solving minimum cost flow problem by successive approximation. *Proceedings of the 19thACM Symposium on the Theory of Computing*, pp. 7-18. Full paper in *Mathematics of Operations Research* 15(1990), 430-466.

[10] STONE, H. S. 1977. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering* 3, 85-93.

[11] AHUJA, R. K., and J. B. ORLIN. 1989. A fast and simple algorithm for the maximum flow problem. *Operations Research* 37, 748-759..

[12] AHUJA, R. K., and J. B. ORLIN. 1989. A fast and simple algorithm for the maximum flow problem. *Operations Research* 37, 748-759.

[13] AHUJA, R. K., and J. B. ORLIN. 1991. Distance-directed augmenting path algorithms for maximum flow and parametric maximum flow problems. *Naval Research Logistics Quarterly* 38, 413-430

[14] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei, "Protocol independent multicast sparse-mode (PIM-SM): Protocol specification." Internet Engineering Task Force (IETF), RFC 2362, June 1998.