Dynamic Mapping of an AS Network into A Smaller Network of Border Routers

Debessay Fesehaye
Department of Computer Science
UIUC, 201 N Goodwin Ave
Urbana, IL 61801-2302, USA
dkassa2@uiuc.edu

***Abstract*—In this paper we present a cross layer routing and congestion control scheme which can map an enterprise network into a smaller network of boarder routers. One of the boarder routers in our scheme called a main node computes the virtual link (tunnel) capacities and corresponding queue size for each path of the ingress routers. This main node can use cloud computing to speed up computation. The scheme can make clean-slate protocols easily deployable in the current Internet with out the need of making changes in the core routers. Besides, the scheme makes online dynamic network diagnosis and analysis easier.**

I. Introduction

Our design of the new scheme which can dynamically map a network of many core routes into a smaller network with only boarder routers is motivated by the following facts. Currently it is difficult to deploy new effective clean slate routing and congestion control protocols by altering the behavior of core routers. Besides, networks usually have few entry points (ingress routers) and few exit points (egress routers). Hence new functionalities can be added to these few entry and exit points either by adding new router-like boxes or leveraging new router functionalities. What goes in via the ingress routers and goes out via the egress routers along with the basic network topology graph can be used to characterize the AS network behavior as shown in the next sections.

II. Mapping the Network

As can be shown in Figure 1, an entire enterprise network can be mapped into a network of its boarder routers and an additional main boarder router. In the figure the letters R, I, E, C and M denote router, ingress, egress, core and main. Our scheme maps the real network graph G_R on the left of Figure 1 to the virtual network graph G_V on the right as follows.

1) Add a new main router node M_1 and connect it to all ingress and egress routers via virtual links.
2) The virtual link(s) connecting each ingress router with each egress router and the M_1 virtual links are paths computed by M_1 for each ingress router and for itself as shown in the next section.

III. Creating the Dynamic Virtual Links (Tunnels)

Each ingress router is connected with its corresponding egress routers via path(s) computed by the main border node M_1. In the mapped virtual network G_R, we call each of these paths a tunnel or a dynamic virtual link (DVL). The main challenges in this mapping are as follows.

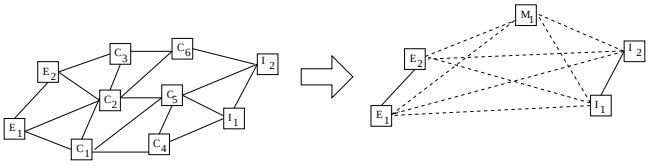

Fig. 1. A Network Mapping

- How to find the dynamic tunnel(s) to connect each ingress router with its corresponding egress router,
- How to obtain the capacity of each dynamic tunnel as each link in the original enterprise network is shared by flows from different ingress routers
- How to find the queue size of each tunnel as packets may be queued up somewhere in the tunnel which connects each ingress with its egress due to bursty nature of network traffic and other discrepancies.

A. How to find the tunnels and their Capacities

Here are the steps the main node M_1 uses to find a tunnel for each ingress router.

- Each ingress router in the enterprise (AS) network sends to the main router node M_1 the rate λ_j at which it is sending packets to an egress router in path j during a control interval d. If we denote the propagation delay of the longest path as ρ, then we can set the control interval d such that $\rho \leq d \leq 2\rho$. The node M_1 knows the links in each path j otherwise it can request it from the ingress routers (to bootstrap).
- The main node M_1 aggregates these values and calculates the fair rate R_i and per packet price (PPP) p_i on the behalf of interface i of each router in the original network G_R as follows:
  The *fairRate* R_i is calculated as
  $$R_i = \frac{C_i - Q_i/d_i}{N_i} \tag{1}$$
  where C_i, Q_i, d_i and N_i are the the capacity of, queue length at, control interval of and number of paths at link i. Here instead of a fair share a weighted proportional share can also be used by associating a weight v_j with each flow j from an egress to an ingress router and assigning $R_j = v_j R_i$ as a share of flow j on link i.
  The rate λ_j at which flow j is sending packets is reduced if the flow is crossing a congested link (shaped or thinned by the bottleneck link). Hence

$$\lambda_j = \begin{cases} \lambda_j & \text{if } \Lambda_i < C_i \\ \frac{\lambda_j}{\Lambda_i} C_i & \text{otherwise} \end{cases} \quad (2)$$

where $\Lambda_i = \sum_j^{N_i} \lambda_j$ is the total load of link i.
Some flows (paths) may not have enough data to send to utilize their share of the bandwidth. This may result in lower link utilization while other legitimate flows which have more data to send could use the bandwidth. To solve this, the main node M_1 counts such flows as partial flows using λ_j/R_i instead of 1.
Hence we need the flow count indicator n_j, which is given by

$$n_j = \begin{cases} 1 & \text{if } \lambda_j > R_i \\ \frac{\lambda_j}{R_i} & \text{otherwise} \end{cases} \quad (3)$$

to find the actual flow count A^i which is then given by

$$A_i = \sum_{j=1}^{N_i} n_j. \quad (4)$$

Therefore the *fairRate* R_i is given as

$$R_i = \frac{(C_i - \frac{Q_i}{d_i})}{A_i}. \quad (5)$$

The λ_j and n_j are obtained by the *aggregator algorithm* at the main node M_1 which is presented in the next section.
The queue rate Q_i/d_i is also given as

$$q_i = \frac{Q_i}{d_i} = \begin{cases} 0 & \text{if } \Lambda_i \leq C_i \\ \Lambda_i - C_i & \text{otherwise.} \end{cases} \quad (6)$$

The total load Λ_i of link i is also obtained using the *aggregator algorithm* at the main node M_1 which is presented in the next section.
Even though more sophisticated pricing functions of the demand can be used to find the PPP, we here use a simple adaptive mechanism.
The unit per packet price (PPP) p_i can be obtained as

$$p_i = p_i^p \frac{R_i^p}{R_i} \quad (7)$$

where p_i^p and R_i^p are the price (PPP) p_i and rate R_i obtained from the previous round (control interval).

- The main node M_1 then calculates the local path, virtual path (tunnel) rate (capacity) C_j and per packet cost of the tunnel P_j for each of its ingress routers.
  - The path is the one with the maximum of the minimum R_i in each path and
  - the cost is the sum of the p_i of each router in the selected path. The *totalPacketPrice* P_j of path j is given by

$$P_j = \sum_i^{\Pi_j} p_i \quad (8)$$

where Π_j is the number of links in path j.

The main node can also use R_i/p_i (high rate and low price) instead of the R_i to find the best path for each ingress router.

B. The Aggregator Algorithm at the Main Node

The main node M_1 computes the fair rate R_i and per packet price (PPP) p_i of each interface i of a router in the AS by aggregating the rate λ_j from the ingress routers as follows:

The main node M_1 first runs a *MaxMin* algorithm like the one in [1] to find the best path set S from each ingress router to the corresponding egress router. During setup the main node M_1 can be configured with a path set S and the corresponding rate λ_j obtained from the ingress routers. Each best path j can be recorded using a structure like

```
typedef struct {
   int pathID;
   list<link> pathLinks;
   int pathHopCnt;
   int pathHopCntr;
   bool pathMarked
} path;
```

where *pathLinks*, *pathHopCnt* and *pathHopCntr* are the list of links, number of hops and current hop counter of the path with *pathID*. The *pathHopCntr* is initialized to 0 and the link structure is as defined below.

While finding the links in each best path j, the main node also counts the in-degree D_i of router i of link i which is the number of paths crossing the link i using a structure like

```
typedef struct {
   int linkID;
   int inDegree;
   set<path> pathSet;
   int markedPathCntr;
   double linkLoad;
   int inDegreeCntr;
   int actualNumFlows;
   bool linkMarked;
   double linkFairRate;
   double linkPrevFairRate;
} link;
```

where *inDegree* and *pathSet* are the number of paths and the set of paths using the link with *linkID*. The *markedPathCntr*, *linkLoad*, *inDegree*, *inDegreeCntr*, *actualNumFLows* and *marked* are initialized to 0, λ_j, 1, 1, 1 and 0 respectively. The aggregation algorithm is then given by Algorithm 1.

C. Co-operation from Egress Routers to find Tunnel Queue Size and Faulty Link

The above aggregator algorithm assumes that if a router or a link in the core network fails, the main node M_1 eventually knows and excludes the link and router from the network when computing a path for the egress routers. However carefully aggregated information from the egress routers along with the

Algorithm 1 The Aggregator Algorithm to calculate R_i and p_i

Require: $S \neq \emptyset$ {Path set is not empty}
  repeat
    for each *path* j in S **do**
      $i \Leftarrow$ current unmarked link in path j
      $I_i \Leftarrow$ *inDegreeCntr* of link i
      $D_i \Leftarrow$ *inDegree* of link i
      if $I_i = D_i$ **then**
        $R_i^p = R_i$ {Previous fair rate = current fair rate}
        Compute q_i using Equation 6
        Compute R_i using Equation 5
        Compute p_i using Equation 7
        Compute λ_j using Equation 2
        Mark link i as done for path j
        Set i to the next link in path j {Next = current}
        $\Lambda_i \Leftarrow \Lambda_i + \lambda_j$ {Load of next link increases by λ_j }
        $I_i \Leftarrow I_i + 1$
        Compute n_j using Equation 3
        Compute $A_i \Leftarrow A_i + n_j$ {Equation 4}
        $h_j \Leftarrow$ hop counter of flow j
        $H_j \Leftarrow$ hop count of flow j
        if $h_j = H_j$ **then**
          Mark path j as completed.
          $m \Leftarrow$ marked path counter
          $M \Leftarrow$ number of paths in S
          $m_i \Leftarrow$ marked link i path counter
          $m \Leftarrow m + 1$
          $m_i \Leftarrow m_i + 1$
          {Reset the hop counter of path j}
          $h_j \Leftarrow 0$
        else
          $h_j \Leftarrow h_j + 1$
        end if
        $M_i = I_i \Leftarrow$ number of paths crossing link i
        if $m_i = M_i$ **then**
          {Reset the additive values}
          $\Lambda_i \Leftarrow 0$
          $A_i \Leftarrow 0$
          $I_i \Leftarrow 0$
        end if
      end if
    end for
  until $m = M$ {Until all paths are traversed}

aggregated information from the ingress routers can also be used to detect anomaly which can be unexpected congestion or link (node) failure and avoid the faulty part of the network. This can be done by comparing the rate λ_j of path j obtained by the aggregator algorithm described above against the actual incoming rate λ_j^e of at the corresponding egress router which is also sent to the main node M_1. If $q_j^e = \lambda_j - \lambda_j^e$ is a big value, then some link or router in path j is faulty. If the values of q_j^e of other paths are also similarly too high, then the link(s) at the intersection of these paths can be the candidates for the faulty link(s). Hence our scheme can also be very useful as for network monitoring and debugging to detect and locate the faulty parts of a network.

After obtaining the maximum R_j of the minimum rates in each path of an AS, the *MaxMin* algorithm can also obtain the capacity C_j by taking the minimum of the R_j and λ_j^e (which is the last rate forwarded from the link of the current AS to another AS). So the C_j considered as the capacity of tunnel j is the maximum of such C_j's computed by taking the minimum of the maximum AS R_j values and the actual λ_j^e value which the corresponding egress measures as having received. This corrects errors caused due unexpected link (node) failure in the AS network.

If the rate R_j of path j as shown above is the maximum of the minimum R_i in each path obtained using the *MaxMin* algorithm, then the real capacity C_j of tunnel j is $R_j - q_j^e$. Hence if there is unexpected queue buildup (or packet drops) somewhere in the tunnel, the capacity of the tunnel becomes very low and hence the path computation algorithm avoids it and chooses the other paths. The path cost can also be similarly adjusted using C_j instead of R_j. The new path price can for instance be given as $\frac{C_j}{R_j} P_j$.

D. Computation as a Map-Reduce Framework

The aggregator algorithm presented above can also be thought of as a Map-Reduce framework where the computations of Λ_i, I_i, A_i and m_i can be done using mappers, the rest of the aggregator algorithm can be done using reducers. The condition $I_i = D_i$ can for instance be considered as a reduce trigger. Further studies can also be done to determine if better data structures and aggregation schemes can be found.

E. Offline Analysis

The path set and the corresponding link values can be sent to an offline analyzer for further data mining and analysis.

IV. Summary

We presented schemes to dynamically monitor and perform online resource allocation on big enterprise networks. This scheme then finds tunnels connecting each ingress router with an egress router and their corresponding dynamic capacities. Our approach can help clean slate protocols to be deployed in the current Internet without requiring changes in the core routers.

References

[1] Fesehaye, D., Gupta, I., and Nahrstedt, K. A Cross-layer Routing and Congestion Control for Distributed Systems. *University of Illinios, Department of Computer Science, Technical Report UIUCDCS-R-2008-3015*, 11 (Apr. 2008).