ECE697J 2002 Project Report
Benchmarking and Simulation of BGP Processing

Jianhong Xia and Jinghua Hu

December 10, 2002

Abstract

This project focuses on the design of benchmarking and simulation environment for BGP processing overhead. We start with the analysis on different cases for BGP UPDATE message handling. Then we design several testbeds for measurements on processing overhead for different procedures. We setup a simple experiment environment and present some preliminary results on BGP processing overhead for starting up, sending UPDATE messages, receiving and processing UPDATE messages. We also suggest some variations on the design of future measurements.

1 Background and Motivation

BGP-4 [1] is a de facto inter-domain routing protocol in current Internet. BGP routers exchange inter-domain routing information with each other by announcing or withdrawing routes to destination prefixes. BGP routers need to process high volume of update messages and calculate best paths accordingly in order to guarantee network connectivity. Overload of CPU in BGP routers may cause instability in routing protocol.

We are concerned about the overhead of BGP processing in current network. Is it possible to measure BGP processing overhead? Are these results helpful for us to design network processors to support BGP protocol? There is no such work done by others so far. In this project, we will work on the analysis and design of benchmark environment for BGP processing.

2 Project Plan

The goal of this project is to investigate the problems on the possibility and requirements of supporting BGP by network processors. We divide it into three stages as below:

1. Design a benchmark for measuring BGP processing overhead.
2. Do measurements and understand BGP processing overhead by simulation.
3. Use the results to obtain design guidelines on supporting BGP protocol by network processors.

To solve this problem with the allocated time, we will focus on the first two stages. The following of this report is organized as follows. In Section 3, we briefly analyze BGP protocol and enumerate basic cases of BGP operations. In Section 4 we present the design of benchmark for measurement on the operation cases. In Section 5, we describe our experimental setup. Some preliminary results will be presented in Section 6. Finally, we will have some discussions on this measurement result and future work.

3 BGP Protocol Analysis

3.1 BGP Message Handling

BGP protocol is transported over TCP. When a BGP-speaking router wants to setup BGP sessions with another BGP router, it will initiate a TCP connection. Once the TCP connection is setup, BGP routers will start exchange information between each other. The processing of each BGP message follows the two stages as below.

- Stage 1: BGP Message Header Parsing
  - Input: BGP message
  - Format:
    - BGP message Header: 19 octets (marker:16, length:2, type:1)
    - BGP message Body: variable
  - Output: BGP message Body
  - Function: Resolve BGP Header, extract length and type, dispatch to corresponding processing unit.
- Stage 2: BGP Message Body Processing
  - Input: BGP message Body Options
    - type=1: OPEN
    - type=2: UPDATE
    - type=3: NOTIFICATION
    - type=4: KEEPALIVE
  - Format: variable
  - Output: variable
  - Function: Validate each attribute fields based on message types, extract useful information and update FSM status or routing database.

3.2 BGP Message Format

Now we look at the format for different types of messages.

- Type 1: OPEN
  - Input: BGP OPEN message. Minimum:10+19=29 octets.
  - Format:
    - Version: 1 octet
    - My AS: 2 octets
    - Hold Time: 2 octets

    - BGP Identifier: 4 octets
    - Opt Parm Len: 1 octets
    - Optional Parm: variable, (type:1, len:1, value:variable)

- Type 2: UPDATE
  - Input: BGP UPDATE message. Minimum:4+19=23 octets.
  - Format:
    - Unfeasible Routes Length: 2 octets
    - Withdraw Routes: variable, N*(length:1, Prefix:variable)
    - Total Path Attr Length: 2 octets
    - Path Attributes: variable, M*(type:2(flag:1+code:1), len:1 or 2, value:variable)
    - Network Layer Reachability Info: variable, K*(length:1, Prefix:variable)
  - Function: Analyze update message, modify FSM status and database.
- Type 3: NOTIFICATION
  - Input: BGP NOTIFICATION message. Minimum: 2+x octets.
  - Format:
    - Error Code: 1 octet
    - Error Subcode: 1 octet
    - Data: variable
  - Function: Handle error messages.
- Type 4: KEEPALIVE
  - Input: None. Already specified by BGP Header.
  - Function: Keep the BGP session alive.

3.3 BGP Operational Cases

We can now enumerate different operational cases for BGP message handling.

- KEEPALIVE
  - BGP Header – KEEPALIVE – Done
- NOTIFICATION
  - BGP Header – NOTIFICATION – Done.
- OPEN
  - BGP Header – OPEN – Option==0 – Done.
  - BGP Header – OPEN – Option!=0 – Process Optional Data – Done.
- UPDATE
  - BGP Header – UPDATE – Withdraw==0 – Announce==0 – Done.

- BGP Header – UPDATE – Withdraw==0 – Announce==1 – Process Announce – Done.
- BGP Header – UPDATE – Withdraw==N – Process Withdraw – Announce==0 – Done.
- BGP Header – UPDATE – Withdraw==N – Process Withdraw – Announce=1 – Process Announce – Done.

In the following part of this report, we will mostly focus on the handling of UPDATE messages because UPDATE messages composite the major part of traffic between BGP routers. Other types of messages are not as important in our experiments because they do not need complicated processing and the volume of those messages are much less than that of UPDATE messages.

3.4 UPDATE Message Handling

3.4.1 UPDATE Message Handling Procedure

Here we first look the procedure for handling UPDATE messages shown in Fig. 1, then we briefly introduce Decision Process which is a key component for the procedure of best path selection.

```
if( the UPDATE Message contains non-empty WITHDRAW field )
    Remove previously advertised routes from Adj-RIB-In
    Run its Decision Process

if( the UPDATE Message contains ANNOUNCEMENT, i.e. a feasible route)
    if( NLRI exists in Adj-RIB-In )
        Replace the older route in Adj-RIB-In
        Run its Decision Process
    elseif( new route is subset of an earlier route in Adj-RIB-In with different path attributes )
        Store the new route
        Run its Decision Process
    elseif( new route is subset of an earlier route in Adj-RIB-In with same path attributes )
        Store the new route (No Decision Process here.)
    elseif( new routes is superset of an earlier route in the Adj-RIB-In )
        Store the new route
        Run its Decision Process
    elseif( new NLRI )
        Store the new route
        Run its Decision Process
```

Figure 1: BGP UPDATE Message Handling Procedure

3.4.2 Decision Process

Decision Process selects routes for subsequent advertisement by applying the policies in the local policy information base (PIB) to the routes stored in its Adj-RIB-In. The output of the Decision Process is the set of routes that will be advertised to all peers; the selected routes will be store in the local speaker's Adj-RIB-Out. There are three phases in Decision Process. Note that DP-Phase 2 and Phase 3 would lock Loc-RIB from each other.

- DP-Phase 1: Calculation of Degree of Preference
  - Matching local import policies to assign LOCAL_PREF (or WEIGHT in CISCO routers). Currently there are two popular MATCH expressions, AS-MAP and IP-PREFIX.
- DP-Phase 2: Route Selection
  - When NEXT-HOP is not reachable, remove route from Loc-RIB and Adj-RIB-In, return.
  - Otherwise, select highest degree of preference of any route to the same set of destinations (Breaking Ties if needed).
- DP-Phase 3: Route Dissemination
  This function shall be invoked on completion of phase 2 or when any of the following events occur:
  - when routes in Loc-RIB to local destinations have changed
  - when locally generated routes learned by means out side of BGP have changed
  - when a new BGP speaker – BGP speaker connection has been established

  This phase is responsible for disseminating routes in the Loc-RIB to each peer located in a neighboring autonomous system, according to the policies contained in PIB.

3.4.3 Update-Send Process

The Update-Send Process is responsible for advertising UPDATE messages to all peers. It includes Internal Updates and External Updates. BGP protocol has constrains on the amount of routing traffic(that is, UPDATE messages) in order to limit both the link bandwidth needed for advertising UPDATE messages and the processing power needed by the Decision Process for digesting the information contained in the UPDATE messages.

4 Testbed Design

4.1 Design Goals

From the above description, we may divide UPDATE message handling in BGP processing into three stages. Note that the third stage does not immediately follow the first two stages, but is triggered periodically by timer.

1. Parse UPDATE messages and Store/Remove/Replace routes into/from Adj-RIB-In
2. Run Decision Process for best path selection, Update database
3. Update-Send Process

Next we will look into a little bit more detail and divide these procedures into some basic units in processing and define the corresponding measurement quantities for benchmarking. The division of the procedures may not be strictly following the implementation of the protocol, but it only serves as our view on basic segments and units in BGP processing.

4.2 Measurement Design

The measurement results on networks with different size/topology may be different. The processing time for one message on a small routing table may be faster than that for the same message on a large routing table due to the database operation overhead.

Currently, the core BGP routers typically have more than 100,000 prefixes in their routing tables. For simplicity, here we can start from a medium size network. We denote N as the number of different prefixes in routing table. After that, we can easily choose different values of N to see how N affects our measurement results.

When looking into more details on the procedure of message handling, we define the following quantities for measurement.

- $T_{withdraw-parse-overhead}$: the time from processing withdraw part to the beginning time of parsing NLRI information
- $T_{withdraw-parse-one-prefix}$: the time for one prefixes in parsing NLRI information of withdraw part
- $T_{announce-parse-overhead}$: the time from processing announcement part to the beginning time of parsing NLRI information
- $T_{announce-parse-one-prefix}$: the time for one prefixes in parsing NLRI information of announcement part
- $T_{insert-one-route}$: the time for inserting a new route in routing table
- $T_{replace-one-route}$: the time for replacing one route in routing table
- $T_{delete-one-route}$: the time for deleting one route in routing table
- $T_{decision-process}$: the time for decision processing

The intuition here is :
$T_{withdraw-parse-overhead} << T_{announce-parse-overhead}$
$T_{withdraw-parse-overhead} << T_{withdraw-parse-one-prefix} + T_{delete-one-route}$
$T_{withdraw-parse-one-prefix} \approx T_{announce-parse-one-prefix}$

Assumptions: We assume that BGP session has already been established, and we only need to feed BGP session update messages(announcement or withdrawal) to measure BGP processing time for these update messages. There is no separate I/O overhead considered in this setting. BGP servers is not overloaded, i.e., there is no drop in the message queue. In most of the following experiments, there is only a single peer session between two directly connected BGP routers.

4.3 UPDATE Message Handling Model

Given the above definitions, we now describe the simplified model for processing BGP UPDATE messages as shown in Fig. 2. This model allows us to come up with the relationship between measurements and design testbed as will be shown in next Section.

```
if( Number of withdrawal is not zero )
    Withdraw-Parse-Overhead
    foreach NLRI
        Withdraw-Parse-One-Prefix
        Delete-One-Route

if( Number of announcement is not zero. Note: this number is either 0 or 1)
    Announcement-Parse-Overhead
    foreach NLRI
        Announcement-Parse-One-Prefix
        if( existing prefixes )
            Replace-One-Route
            Decision-Process
        elseif( new route is subset of an earlier route with same path attributes)
            Insert-One-Route
        else
            Insert-One-Route
            Decision-Process
```

Figure 2: BGP UPDATE Message Handling Model

4.4 TestBed Design

4.4.1 Measurement on Withdraw Process

1. Testbed A:

    - step 1: build initial routing table for 10,000 unique prefixes
    - step 2: feed 10,000 withdraw messages, each withdrawing one prefix.

    We measure T_a as the active running time of step 2. In this case, when BGP withdraws any route, the prefix will not exist in the routing table. So the Decision Process will be finished quickly because there is no more routes in routing table. Thus here we ignore the processing time for Decision Process.

    $T_a = 10,000 * (T_{withdraw-parse-overhead} + 1 * (T_{withdraw-parse-one-prefix} + T_{delete-one-route}))$

2. Testbed B:

    - step 1: build initial routing table for 10,000 unique prefixes
    - step 2: feed 100 withdraw messages, each withdrawing 100 prefixes.

    we only measure T_b as the time of step 2. In this case, when BGP withdraws any route, the prefix will not exist in the routing table. So the Decision Process will be finished quickly because of no routes in routing table. We ignore the Decision Processing here.

    $T_b = 100 * (T_{withdraw-parse-overhead} + 100 * (T_{withdraw-parse-one-prefix} + T_{delete-one-route}))$

From Testbed A and B, we can get:
$T_{withdraw-parse-overhead} = (T_a - T_b)/9900$
$T_{withdraw-parse-one-prefix} + T_{delete-one-route} = (T_b * 100 - T_a)/990000$

4.4.2 Measurement on New Announcements

1. Testbed C:

   - step 1: feed 10,000 announcement messages, each message announces one unique prefix. In this case, each route will be new in the routing table.

   We measure T_c as the running time of this step.
   $T_c = 10,000*(T_{announce-parse-overhead}+T_{announce-parse-one-prefix}+T_{insert-one-route}+T_{decision-process})$

2. Testbed D:

   - step 1: feed 100 announcement messages, each message announces 100 unique prefixes. In this case, each route will be new in the routing table.

   We measure T_d as the running time.
   $T_d = 100*(T_{announce-parse-overhead}+100*(T_{announce-parse-one-prefix}+T_{insert-one-route}+T_{decision-process}))$

From Testbed C and D, we can get:
$T_{announce-parse-overhead} = (T_c - T_d)/9900$
$T_{announce-parse-one-prefix} + T_{insert-one-route} + T_{decision-prcoess} = (T_d * 100 - T_c)/990000$

4.4.3 Measurement on Decision Process

1. Testbed E:

   - step 1: build initial routing table for 10,000 unique prefixes
   - step 2: feed 10,000 announcement messages, each message announces one prefix. those 10,000 prefixes are the subset of existing prefixes, and the routes have the same path attributes.

   we measure T_e as the time for step 2. In this case, BGP will not do decision processing.

   $T_e = 10,000 * (T_{announce-parse-overhead} + T_{announce-parse-one-prefix} + T_{insert-one-route})$

2. Testbed F:

   - step 1: build initial routing table for 10,000 unique prefixes
   - step 2: feed 100 announcement messages, each message announces 100 prefixes. those 100 prefixes are the subset of existing prefixes, the routes have the same path attributes.

   we only measure T_f as the time of step 2. In this case, BGP will not do decision processing.

   $T_f = 100 * (T_{announce-parse-overhead} + 100 * (T_{announce-parse-one-prefix} + T_{insert-one-route}))$

From Testbed E and F, we can get:
$T_{announce-parse-overhead} = (T_e - T_f)/9900$
$T_{announce-parse-one-prefix} + T_{insert-one-route} = (T_f * 100 - T_e)/990000$
$T_{decision-process} = (T_d * 100 - T_c)/990000 - (T_f * 100 - T_e)/990000$
(Here decision processing is based on single available route for each prefix)

4.4.4 Measurements on Route Replacement

1. Testbed G:

   - step 1: build initial routing table for 10,000 unique prefixes
   - step 2: feed 10,000 announcement messages, each message announces one prefix. those 10,000 prefixes are the existing prefixes, but the routes have different path attributes.

   we measure T_g as the time for step 2. In this case, BGP will do decision processing.
   $T_g = 10,000*(T_{announce-parse-overhead}+T_{announce-parse-one-prefix}+T_{replace-one-route}+T_{decision-process})$

2. Testbed H:

   - step 1: build initial routing table for 10,000 unique prefixes
   - step 2: feed 100 announcement messages, each message announces 100 prefixes. those 100 prefixes are the existing prefixes, but the routes have different path attributes.

   we only measure T_h as the time of step 2. In this case, BGP will do decision processing.
   $T_h = 100*(T_{announce-parse-overhead}+100*(T_{announce-parse-one-prefix}+T_{replace-one-route}+T_{decision_process}))$

From Testbed G and H, we can get:
$T_{announce-parse-overhead} = (T_g - T_h)/9900$
$T_{announce-parse-one-prefix} + T_{replace-one-route} = (T_h * 100 - T_g)/990000 - T_{decision-process}$

4.4.5 Other Measurements

1. Testbed I: Mixed withdraw and announcement The main purpose for this testbed is to verify the previous division and measurements of individual components.

   - step 1: build initial routing table for 10,000 unique prefixes
   - step 2: feed 100 announcement, each message withdraw 50 prefixes and announce 50 new subset prefixes with same path attributes.

   we measure T_i as the time for step 2.
   $T_i = 100*(T_{withdraw-parse-overhead}+50*(T_{withdraw-parse-one-prefix}+T_{delete-one-route})+T_{announce-parse-overhead}+ 50 * (T_{announce-parse-one-prefix} + T_{insert-one-route}))$

2. Testbed J: Decision Processing for large routing tables and multiple available routes
   Suppose we are interested in decision processing in N prefix entries routing table, and each prefix has M available alternative routes. Here we define N=10,000 and M = 10.

Depending on the sequence of announcement messages, Decision process may spend very long or very short time. Here we define two extreme cases and one general case.

- case 1: for each prefix, the route of later announcement will supersede the former to be the best path.
- case 2: for each prefix, earliest route is the best route among all announcement.
- case 3: for each prefix, the best path is randomly picked up among all available routes.

We can design these routes for each prefix have different AS-PATH length. BGP router will always choose the shortest path as the best route. So we see that in case 1, the path length of each announcing route will be in decreasing order of the sequences of update messages. in case 2, the path length of each announcing route will be in increasing order of the sequences of update messages. in case 3, the path length of each announcing route is randomly among all update messages.The intuition here is:
$T_{decision-processing-case-1} > T_{decision-processing-case-3} > T_{decision-processing-case-2}$

4.4.6 Testbed Summary

From the above testbed plan, we see that if our analysis correctly reflect the real processing, and if these experiments can be performed properly, we have the following possibly measurable quantities:

- $T_{withdraw-parse-overhead}$
- $T_{withdraw-parse-one-prefix} + T_{delete-one-route}$
- $T_{announce-parse-overhead}$
- $T_{announce-parse-one-prefix} + T_{insert-one-route}$
- $T_{announce-parse-one-prefix} + T_{replace-one-route}$
- $T_{decision-process}$

Note that there are still some quantities can not be measured individually.

5 Experimental Setup

Note: Due to the time limit for this project, in this report we only try the following simplified experiments for BGP processing overhead measurement. Only when the result verifies our assumptions can we continue on with our complicated Testbed experiment plan presented in Section 4.

5.1 Zebra and BGPd

In order to understand the behavior of BGP protocol processing and we may need CISCO routers or routing software to build our testbed. Since CISCO BGP routers are too expensive and not flexible to setup the required topology for our testbed, we choose to use routing software to simulate BGP router.

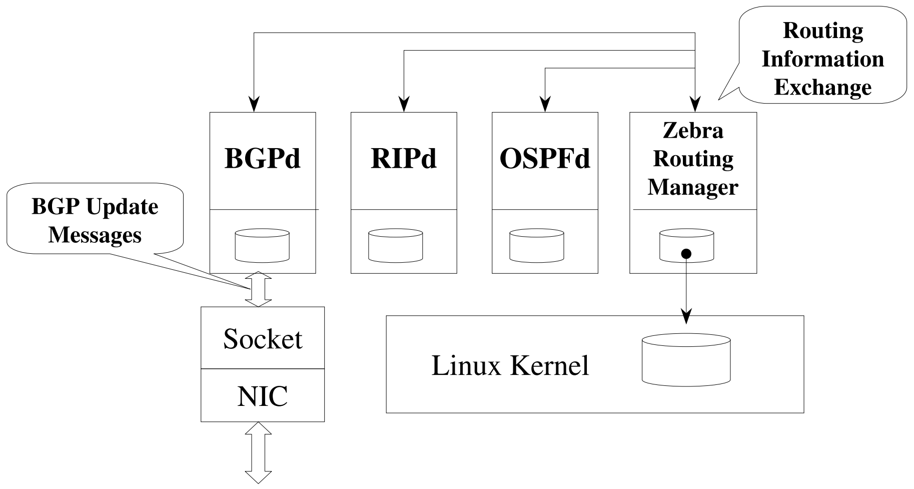


Figure 3: Operation Model of Zebra

There are several routing softwares supporting BGP protocol, such as Zebra [2], GateD and MRTd. More and more researchers like to use Zebra because it is free, easy to setup and configure and it has a similar interface as CISCO's command style configuration interface.

GNU Zebra is a free routing software (distributed under GNU Generic Public License). It supports BGP-4 protocol as described in RFC1771 (A Border Gateway Protocol BGP-4) and other popular routing protocols. Zebra software offers true modularity in its software design. It uses multi-thread technology. With the support of multi-threading UNIX/LINUX kernels, there is one process corresponding to each protocol. Thus Zebra provides both flexibility and reliability. Each module can be upgraded and working independently from the others.

In Zebra software, there are four Daemons running: BGPd, RIPd/RIPngd, OSPFd/OSPF6d and Zebra. BGPd Daemon manages BGP-4 and BGP-4+ protocol. RIPd/RIPngd Manages RIPv1/RIPv2/RIPng protocol. OSPFd/OSPF6d Manages OSPFv2/OSPFv3 protocol. All these protocol Daemons are handling their own protocol only. Zebra Daemon provides kernel routing table update and coordinates protocol daemons to redistribute routing information among them. The operation model of Zebra is illustrated in Fig. 3.

5.2 Running Time Measurement Tool

We choose "time" command in Linux OS as the tool for rough measurements on BGP update processing overhead. The output of this command will give the value of real running time, user space time and system space time. The accuracy of the output time is 10^-3 second. We will mainly look at the total time from user space and system space time, i.e., the time we are measuring is:
$T_{total-time} = T_{user-space-time} + T_{system-space-time}$.
We will also look at relative processor share for user space time:
$proc-share-user = T_{user-space-time}/T_{total-time} = T_{user-space-time}/(T_{user-space-time}+T_{system-space-time})$.

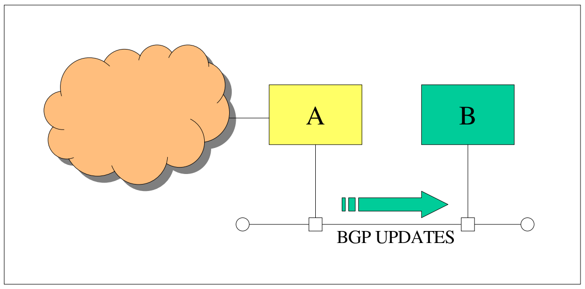


Figure 4: Testbed Topology

5.3 Topology and Hardware Setting

The experiment environment is shown in Fig. 4. We use two Linux Workstations for this experiments. We refer them as Workstation A and Workstation B. In the following experiments, A will take care of the initiation of advertisement of UPDATE messages, and B works passively, receiving and handling the incoming UPDATE messages. So we can roughly say that A is the sender, and B is the receiver. Here are the system setting for these two workstations:

- Workstation A: Intel Pentium III 866MHz, 133 front side bus, 256K full speed cache, RIMM 256MB PC800 at 400MHZ.
- Workstation B: Intel Pentium III 800MHz, 133 front side bus, 256K full speed cache, RIMM 256MB PC600 at 300MHZ.

5.4 Simplified Measurements

Our first step is to get a proof of our idea whether the processing overhead of BGP protocol is measurable with this experiment setting. So we start from a much simplified testbed with another set of measurements defined as below.

- $T_{startup-overhead}$: the time for BGP to startup
- $T_{startup-and-send-all}$: the time for BGP to startup and send out all announcements
- $T_{send-update-all}$: the time for BGP to send out announcements for all prefixes
- $T_{receive-update-all}$: the time for BGP to receive/process all incoming UPDATE messages

Note: $T_{startup-and-send-all} \approx T_{startup-overhead} + T_{send-update-all}$.
Processing overhead per prefix will be estimated from the total running time in each cases across different routing table sizes.

5.5 Configuration files

We plan to measure BGP processing overhead in networks with different size. For sender (Workstation A), we prepared six different configuration files for initialization of routing table. These files represent the setting for different routing table size. Total numbers of reachable prefixes contained in the files range from 1, 10, 100, up to 100,000.

For receiver (Workstation B), we start from an empty routing table, so the startup time for B is ignored. We configure B to only receive UPDATE messages from A and not send out any advertisements to other BGP routers.

For each of the above different settings, we look at the processing overhead for BGP to startup, to send out all UPDATE messages, to receive and process all incoming UPDATE messages. During the running of BGPd, we turn off all logging/debugging information output to reduce the interference.

5.6 Test scenarios

We've designed the following two sets of test scenarios. For each set of configuration files, and for each of these experiments, we repeat 10 times.

1. Set I: Measurement on BGP startup time
   This set of experiments only requires A to be running. We start BGPd with different initial configurations, let it run for a time long enough for startup, then turn it off. The purpose is to measure the startup overhead for BGP routers due to the loading of routing table. The running time for this setting is measured as: $T_{startup-overhead}$, which is time for BGP to load all path information from configuration file into routing table.

2. Set II: Measurement on processing time for Sending/Receiving Announcements
   In this set of experiments, we measure time from both A and B. We first start A with different initial configurations. Then we start B an empty routing table. When the BGP session between A and B is setup, A starts advertising of the whole routing table to B. After all the UPDATE messages have been processed by B, we first turn off B, then turn off A.

   Running time measured from A is the total time of BGP startup and sending out all UPDATE information: $T_{startup-and-send-all} \approx T_{startup-overhead} + T_{send-update-all}$.
   Here it is difficult to split the two terms in the righthand side of the equation in our experiments. So we subtract the mean value of $T_{startup-overhead}$ estimated from experiments Set I to get the estimation of $T_{send-update-all}$ as:
   $T_{send-update-all} \approx T_{startup-and-send-all} - avg(T_{startup-overhead})$.
   Since B starts with an empty routing table, we can basically ignore the startup time for B. So the running time measured from B is approximately the total time for B to receive and process all incoming UPDATE messages: $T_{receive-update-all}$.

6 Preliminary Results

From the above experiments, we are able to show the measurement results for four quantities. Three measured quantities include $T_{startup-overhead}$, $T_{startup-and-send-all}$, $T_{receive-update-all}$, and one estimated quantity is $T_{send-update-all}$.

	startup		startup+sending		receiving	
Number of prefixes	mean	std	mean	std	mean	std
100,000	9.3200	0.2062	16.1250	0.3449	6.1790	0.1588
10,000	1.4000	0.0346	1.6220	0.0377	0.6580	0.0483
1,000	0.1780	0.0132	0.2040	0.0117	0.1110	0.0099
100	0.0500	0.0067	0.0540	0.0070	0.0590	0.0074
10	0.0390	0.0057	0.0420	0.0063	0.0530	0.0067
1	0.0370	0.0067	0.0380	0.0079	0.0580	0.0063

Table 1: Statistics on Measurement Result.

6.1 Overview on Sample Data

The left column in Fig. 5 shows the sample data point obtained from our measurements. The right column in the figure shows the processor share of user space time as the percentage of total running time. Statistical results for these sample data is listed in Table. 1 where Mean values and standard deviations are shown in separate columns for each set of measurements.

From Fig. 5 and Table. 1, we get the following observations:

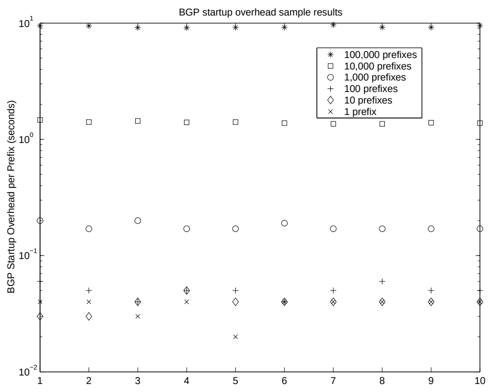
1. For the data sampled at different runs, the processing time measured by "time" command shows better consistency in the cases when the number of prefixes is large (e.g., when number is larger than 100) than the cases when the number of prefixes is small. This suggests that it is possible to get measurements on BGP processing overhead by the simple experimental settings. Since we have ignored many factors in our assumptions, the results shown in the cases with smaller routing table may get more noise introduced. So in the following part, we will do comparisons mostly on the cases with medium to large routing tables ($N = 100$ and above).

2. From the processor share of user space time chart, we see that for the overhead of BGP pure startup and BGP startup+sending, the user space time consumes more than 80% of the total time. In the case of BGP receiving-UPDATE, the user space time consumes about 50% to 80% of the total time. This suggests that in the processing of incoming UPDATE messages, BGP has more chances to access kernel space than in the other two cases.

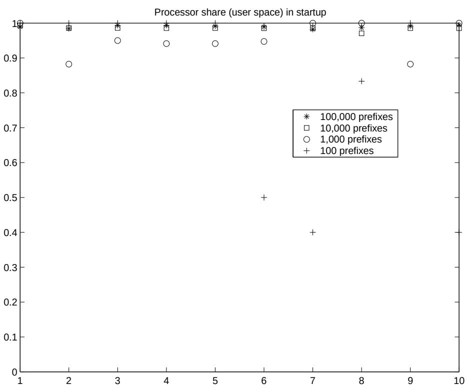
6.2 BGP processing overhead comparison

Fig. 6 is the comparison of BGP processing overhead over different routing table sizes and different scenarios. The overall processing times in Fig. 6(a) are measured/estimated from the above experiments. Here we treat the measurements for $N = 1, 10$ as the common overhead for all testing cases. We subtract this common overhead from all measurements, and divide the resulting time by the number of prefixes. This way, we get an estimation on processing time per-prefix in Fig. 6(b). We have the following observations from the graphs:

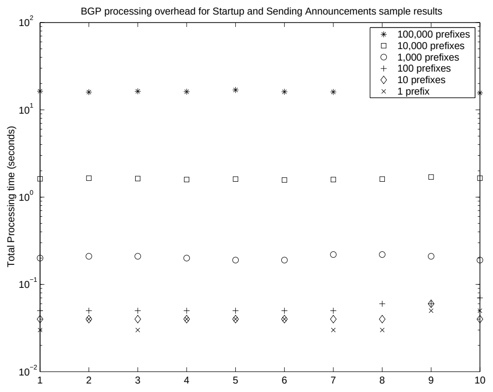
1. Time scale for per-prefix processing
   Approximate per-prefix processing time in startup:
   $T_{startup-overhead-one-prefix} \approx 1.0 \sim 1.5 * 10^{-4} sec.$
   Approximate per-prefix processing time for sending out UPDATE messages:



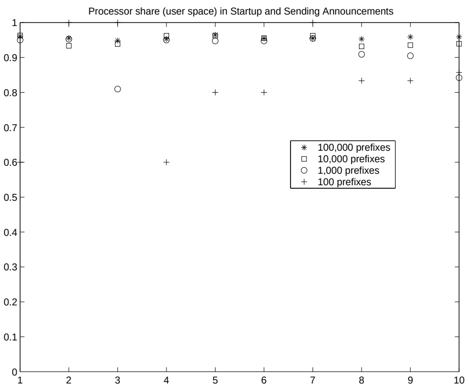
(a) Sample Data for BGP Startup



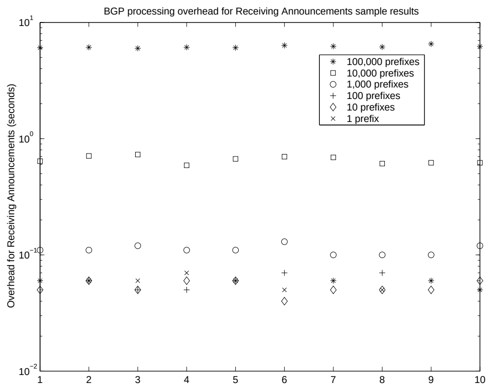
(b)Processor share for user space in Startup



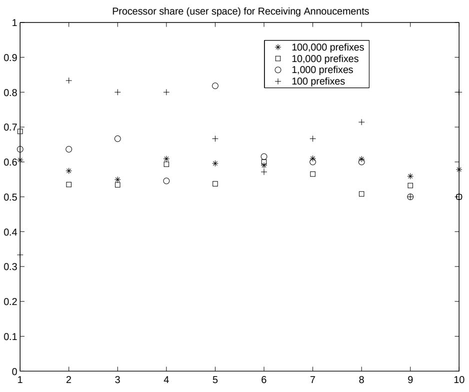
(c) Sample Data for Startup+Sending



(d)Processor share in Startup+Sending

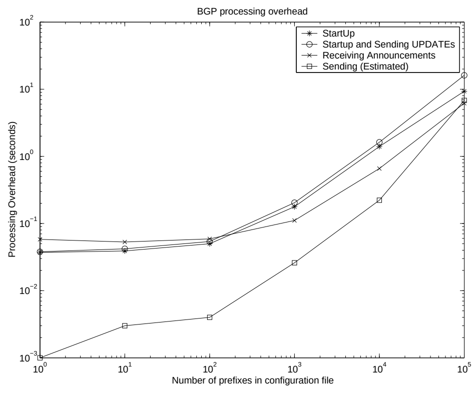


(e) Sample Data for Receiving UPDATEs

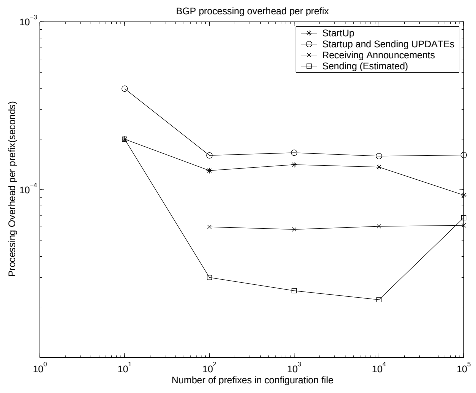


(f)Processor share in Receiving UPDATEs

Figure 5: Sample Data and Processor Share for user space time



(a) BGP processing overhead: Total time



(b)BGP processing overhead: Per Prefix

Figure 6: Comparison of BGP processing overhead

$T_{send-update-one-prefix} \approx 0.2 \sim 0.3 * 10^{-4} sec$
Approximate per-prefix processing time for receiving and processing incoming UPDATE messages:
$T_{receive-update-one-prefix} \approx 0.5 \sim 0.6 * 10^{-4} sec.$

2. Total processing time nearly proportional to routing table size
   For each of the four test cases, the processing overhead for medium to large size routing table is roughly proportional to the number of prefixes processed, which corresponds to the linear part of the lines in part (a) and the consistency in the processing time per prefix across different table sizes.

3. Relative time scale for different procedures
   When looking at the variations across the four different procedures, we get a rough idea on the relative scale between the different types of overhead for medium size routing table. We have:
   $T_{startup-overhead} > T_{receive-upate-all} > T_{send-update-all}$. and for per-prefix processing,
   $T_{startup-overhead-one-prefix}/T_{receive-upate-one-prefix} \approx T_{receive-upate-one-prefix}/T_{send-update-one-prefix} \approx 2.0 \sim 3.0$.
   An exception comes from the case when the number of prefixes is 100K. In this case, processing overhead per prefix for sending UPDATEs is even larger than the receiving process. We are not clear about the reason yet.

7 Discussions and Future Work

7.1 Difficulties

During the analysis and design of BGP benchmark, we have the following major concerns that we are still not sure about the outcome. There are some difficulties that may restrict the actual experiment setup in contrast to our idealized environment.

1. Difficulties in accurate measurement of time.
   We've seen that for smaller routing tables, the measurement sample points does not show consistency due to the limited resolution of "time" command measurement. Although "time" is good

enough for getting a rough view on BGP processing, we may need other tools for better measurements and understanding on running time.

2. Difficulties in control BGP update messages generation.
   Our testbed requires us to have control over the number of update routes sent in a single message. In real environment, it may be very difficult to realize. The reason is that BGP messages are carried by TCP transportation. To reduce the total number of UPDATE messages needed for the advertisement of the whole routing table, currently BGP implementation may choose to fit as many UPDATE routes into a single packet as long as the size does not exceed the maximum limit. So it might be very hard for us to take control on the number of BGP update routes carried in a single UPDATE message. We may need more manipulation on BGP routing tables to take full control.

3. Difficulties in determining the completion of the processing
   To reduce possible interfere and extra processing cost, in our experiments, we need to turn off all logging/debugging information. So in the cases when we need to observe the behavior of a large size routing table, we have to wait long enough time to ensure the completion of processing of all received UPDATE messages before we turn off BGP session.

4. Noisy system behavior
   In our measurements, we have made a lot of assumptions and approximations. In order to get good measurement, we may need to reduce system "noise" as much as possible. Sometimes this is hard to do because BGP needs to do periodical maintenance on its timers and FSM.

7.2 More Variations on Measurements

The preliminary results we obtained are only a coarse-grained view on the BGP protocol. We have thought about the following variations in test setting and implementation that could affect the measurement results.

1. Order of incoming UPDATE messages
   We have seen from the analysis of UPDATE message handling that this procedure mainly consists of the operations (insertion/deletion/update/query) on routing database, and we notice that computation complexity may vary with the incoming data.

   We are not clear about the internal data Structure for storage and processing on the three databases involved in BGP processing. We suspect that not only the number of UPDATE messages, but also the order of the messages may affect the processing time. Currently in our configuration files, all the prefixes are /24 addresses generated automatically in basically a linear increasing order. We may choose to feed the messages in random order to see the effects.

2. Number of BGP peers
   Number of active BGP sessions connected between the receiver of UPDATE messages and BGP peers is another factor. The source of incoming UPDATE messages may affect the processing complexity of Decision Process, especially for the segment of best path selection. There are basically three different cases for message sources:

   - Single Peer: In this case, there is only one available path learned from the peer for each reachable prefix. The decision process is very simple because it has no more choices.

- Multiple Peers located in a single AS: In this case, there are multiple available paths for each prefix, while they are all announced by a single AS. The Decision Process has to choose from the multiple paths for the best one.
- Multiple Peers located in multiple ASes: In this case, there are multiple available paths for each prefix, while they may be announced from different ASes. The Decision Process here has to first find the best-candidate for each source AS, then determine the overall best path from these candidates. This procedure is more complex than case 1, and may need more processing time.

3. Route Aggregation
   Route aggregation may also affect processing overhead because of the change in entries in routing database. But in our testbed design, we avoid aggregation by generating configuration files in such a way that prefixes are all isolated address blocks and can not be aggregated.

7.3 Future Work

For our future work, we will try to overcome the difficulties we mentioned and do measurements on the testbed we suggested in Section 4. Based on these measurements, we will also check the possibility of isolating Zebra/BGPd for processor simulation.

References

[1] Y. Rekhter and T. Li. A border gateway protocol 4 (BGP-4). Request for Comments 1771, March 1995.

[2] http://www.zebra.org/.