MV learning Zhe Song’s code

July/Aug 2014

Input data folder: For each AS folder

1. AR-BGP1-AS1: To configure AR-BGP: transit-or-stub; local preference for each neighbor; MED for multiple boundary nodes to same neighbor AS - Currently not being read by any code
2. Topology AS1: Format – files are being read by main.cpp

AS number

Number of nodes in AS

Node1 Node2 Link-Weight Link-Rate (for intra-domain links)

[MV: Aug. 12: these lines should go away after move to Inter-AS-Info.txt]

0 in a line is just a separator between intra- and inter-domain links

For inter-domain links: AS number: node number for each end of link; weight and rate

Common folder

1. common\_parameter – file being read by CallGenerator.cpp; it should also be read by main.cpp to get extension (max call duration) has comments embedded to describe USST and EST ratio and parameters. //USST parameter: rate(Gbps) duration(minutes) AR\_options call\_fraction: Used by CallGenerator. All USST calls are assigned the same rate and duration. //EST parameter: rate(Gbps) duration\_min(minutes) duration\_max(minutes) AR\_options call\_fraction zipf\_alpha. The call\_fraction selects the percent of calls that should USST. AR\_Options for USST is the number of options to be selected at random assuming uniform distribution from the whole AR window. For EST calls, the AR\_Options is the number of options. But all calls will choose 1, 2, 3 – which are interpreted as start-time options past the lead time. Zipf is used between duration\_min and max with parameter zipf\_alpha. All EST calls have same rate. Change name of this file to input\_parameters\_for\_call\_gen.txt. Add call arrival rate to this file. Add src\_dst\_prob\_matrix\_selector. This can take on three values now: 1: two-level uniform; 2:intra-domain-only; 3: node-level uniform. By having just one file, the same set of parameters are being used for calls that are generated for each AS. If we want to run a simulation with additional calls, we need to write a separate input file and separate module. Alternative would be to require one of these files for each AS, which seems excessive.
2. Create another file called AR\_system\_input\_parameters.txt (file is not yet there and so no one is reading it: CallGenerator and ARServer: Duration\_of\_timeslot\_in\_seconds, AR\_Window\_in\_seconds, Lead\_Time\_in\_timeslots

Explanation for how to select values and how to interpret call-arrival time: How to pick Duration\_of\_timeslot\_in\_seconds: See paper A\_l (t): available bandwidth per timeslot. If our AR\_Window is 24 hours, this means it is 86400 seconds. If we make Duration\_of\_timeslot equal to 1 sec, which means Duration\_of\_timeslot\_in\_seconds equals 1, then the A\_l(t) would need 86400 entries. This will slow down the computation. Therefore let us use 10sec for a timeslot. Earlier we had chosen 1 min or 60 as the value for Duration\_of\_timeslot\_in\_seconds. But for EST calls with 75 GB as min. value and 10 Gbps as rate, the min. duration is only 1 min. So if we make the Lead\_Time\_in\_timeslots 3, then the waiting time is minimally 3 mins when transfer time is only 1 min. So we decided to keep 75 GB as min. value, 10Gbps as link capacity, and hence lowered Duration\_of\_timeslot to 10 sec. A\_l(t) will need a vector of 8640 entries. AR\_Window is 86400 secs (24 hours – which sounds low because Computation scheduling often requires days notice). Lead\_Time still needs to be minimally 1 because a call may arrive at the end of a 10-sec timeslot. Which means if we need 1 sec for round-trip prop. delay for Create-Res message to travel to all PCEs on path, then there will always be a min. 10sec overhead. This is the reason for pushing us to make timeslot duration be 1sec. But the computation argument wins here.

[Prop. delay is max of 500ms-600ms]. So assume that in 1 to 1.2 sec, round-trip Create-Reservation procedure can complete. So lead\_time should be at least say 1.2 sec. So now by choosing lead-time to be 1 TS and making 1 TS equal 10 sec, we have lots of buffer to be sure that create-reservation will have completed). Extension unit is seconds.

Now finally call arrival times are continuous variables – interpreted in seconds. And so the arrival instant is mapped to one 10-sec TS.

So typical values for this AR\_system\_input\_parameters.txt: Duration\_of\_timeslot\_in\_seconds = 10, AR\_Window\_in\_seconds = 86400, Lead\_Time\_in\_timeslots = 1.

1. common/Inter-AS-Info.txt: main.cpp needs to read this file This file has number of ASes, and for each pair of AS in each direction, it gives propagation delay between ARServers. The inter-AS link information was spread in the AS input files where each gives its own AS information. But there was duplication of information and we right away found different weights for 1:1 4:8 in AS1 vs 4:8 1:1 in AS4. So bring that information about inter-AS links into this file.

Src files:

First set:

randgen.c, randgen.h: Original obtained from Web site and Zhe modified. Contains functions to generate random samples drawn from Zipf, Exponential, categorical (generalized Bernoulli) and uniform.

Test these functions by writing your own main.cpp and calling these functions and understanding the output. Use R histograms to plot output to see if the distributions of the samples mirror the underlying population distribution. Run goodness-of-fit tests and qqplots.

Second set:

CallGenerator.cpp and CallGenerator.h.

Zhe created folders under Dropbox\AAAMEDPaperProject\Inter-AS-AR-Simulator\test for unit testing. He created a testCallGenerator program. This Makefile calls and uses files in the src directory such as CallGenerator.cpp and .h. It has to be called with several arguments.

**Third set:**

src\_dst\_prob\_matrix\_generator: Elahe wrote this program to generate three types of matrices: 1: two-level uniform; 2:intra-domain-only; 3: node-level uniform

This program may run separately just to create matrices. During actual simulation, CallGenerator will read appropriate matrix based on selector value specified by user.

**Fourth set:**

ARBGP: In the simulation there will be as many ARServer instances as there are ASes. main.cpp creates these instances. See ARServer.h file. Within the ARServer class, there is an ARBGP public object.

The functions in ARBGP are to createupdate, parseupdate

**Fifth set:**

ARServer: object is created by main.cpp with three arguments: ASnumber, AR\_Window and Extension. Extension is max call duration. So main.cpp should read input\_parameters\_for\_call\_gen.txt to determine maximum call duration and then find Extension as max of USST and EST calls. It is called extension because if the random start-time generator chooses the last option, then the system should track available BW for AR-window + extension. Also main.cpp should read AR\_system\_input\_parameters.txt to get AR window size, which it needs to call ARServer. To get the first parameter num-of-AS it needs to read Inter-AS-Info.txt

**Sixth set:**

IPCE

**Seventh:**

main.cpp: see changes mentioned in ARServer