Programming Assignments for Graduate Students using GENI

Named Data Networking

Please direct comments regarding this document to `fahmy@cs.purdue.edu`.

1 Introduction

This project leverages resources on the ProtoGENI aggregate, using the Omni [6] and Flack [4] GENI clients. The ProtoGENI tutorial [9] is a good starting point to become familiar with the ProtoGENI aggregate. General documentation on the GENI project and its available resources is found on the GENI wiki [7].

GENI resources are shared resources provided by members of the networking community. Please release your slivers when you are done with them, or if you are going to have to leave them for an extended period of time. Remember that in many cases you may be able to perform an experiment, copy the data off to another host, and release the sliver.

1.1 Objectives

The objective of this assignment is to familiarize you with named data (sometimes called content-centric) networking. You will specifically learn about:

- The Project CCNx [5] named data networking overlay for IP networks
- Architectural features of CCNx and other content-centric networking, such as publish-subscribe content distribution, caching at intermediate nodes, and intelligent request forwarding
- Using named data networking (NDN) to fetch content from multiple sources
- The effects of cache granularity on bandwidth requirements for content distribution in a named data network
- The usage of intelligent forwarding to reduce bandwidth requirements in a collaborative environment

1.2 Assignment Material

The material for this assignment can be found at:
`http://www.cs.purdue.edu/homes/fahmy/geni/geni-ndn.tar.gz`

1.3 Tools

ccnd The `ccnd` daemon and its associated software is the reference implementation for the content-centric networking protocol proposed by Project CCNx [5]. This assignment uses release 0.6.1 of the `ccnx` software package. More information on this package can be found at the Project CCNx web site, `http://www.ccnx.org/`. The CCN protocol embodied in CCNx is an attempt at a clean-slate architectural redesign of the Internet.

Atmos The Atmos software package from the NetSec group at Colorado State University [8] will provide named data for your network in the form of precipitation measurements. The Atmos tools are installed in `/opt/ccnx-atmos`.

NetCDF Tools NetCDF (Network Common Data Form) is "a set of software libraries and self-describing, machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data." The NetCDF tools package provides several tools for manipulating NetCDF data, including `ncks`, a tool for manipulating and creating NetCDF data files; `ncdump`, a similar tool used for extracting data from NetCDF data files; and `ncrcat`, a tool for concatenating records from multiple NetCDF data files.

Flack Flack [4] is a graphical interface for viewing and manipulating GENI resources. It is available on the web at `http://www.protogeni.net/trac/protogeni/wiki/Flack`. You may use either Omni [6] or Flack to create and manipulate your slices and slivers, but there are certain operations for which you will probably find one or the other easier to use. You will have to use the Flack client to *Instrumentize* your slices, as you will be directed some exercises.

2 Setting up `ccnd`

The `ccnd` package will be installed on each node allocated by the RSpecs provided with this assignment, and the `ccnd` daemon itself will be started at boot time. The CCNx binaries will be added to the default system shell paths by default, but if you use an alternative shell you may wish to add the directory `/opt/ccnx-0.6.1/bin` to your `PATH` environment variable to save some typing. On any machine from which you need to manipulate the `ccnd` network, you must create keys for your user. In order to do this, log into the host normally via `ssh` and execute the `ccninitkeystore` command from the CCNx software package.

3 Exercises

3.1 Routing in a Named Data Network

In this exercise, you will build a named data network using the Project CCNx [5] networking daemon `ccnd` and transfer scientific data in the NetCDF [3] format between data providers and data consumers using routing based on content names. For the purpose of this exercise, the network will use static routes installed at slice setup. Content will be served and consumed by the Atmos software package.

Create a GENI experiment using the `precip.rspec` file in the provided materials. It will allocate a network such as that depicted in Fig. 1, with `ccnd` running on every host and an Atmos server running on the hosts `datastore1` and `datastore2`.

Routing in named data networks takes place using the namespace under which data is stored. In the case of CCNx, the data namespace is a URI [2, 1], so it looks similar to the familiar `http://` and `ftp://` URLs of the World Wide Web. CCNx URIs are of the form `ccnx:<path>`, where `<path>` is a typical URI slash-delimited path. The components of the URI path are used by the network to route requests for data from consumers to the providers of that data. Routes have been installed on all of the machines in the network except `researcher2`, a data consumer node with two connections to the rest of the named data network.

TASK 1.1: Finalize Configuration

In this task, you will observe the effects of the missing static routes, and restore them.

Log into the host `researcher1`. Run the command `ccninitkeystore`. Issue the command `ccndstatus` to print the status of the `ccnd` daemon on the local host. Note the number of content items stored on this host (which should be zero, if you have not yet fetched any content) and the number of interests accepted (under “Interest totals”).

On `researcher1`, issue a request for precipitation data from 1902/01/01 through 1902/01/02. To do this, use the Atmos tools by invoking:

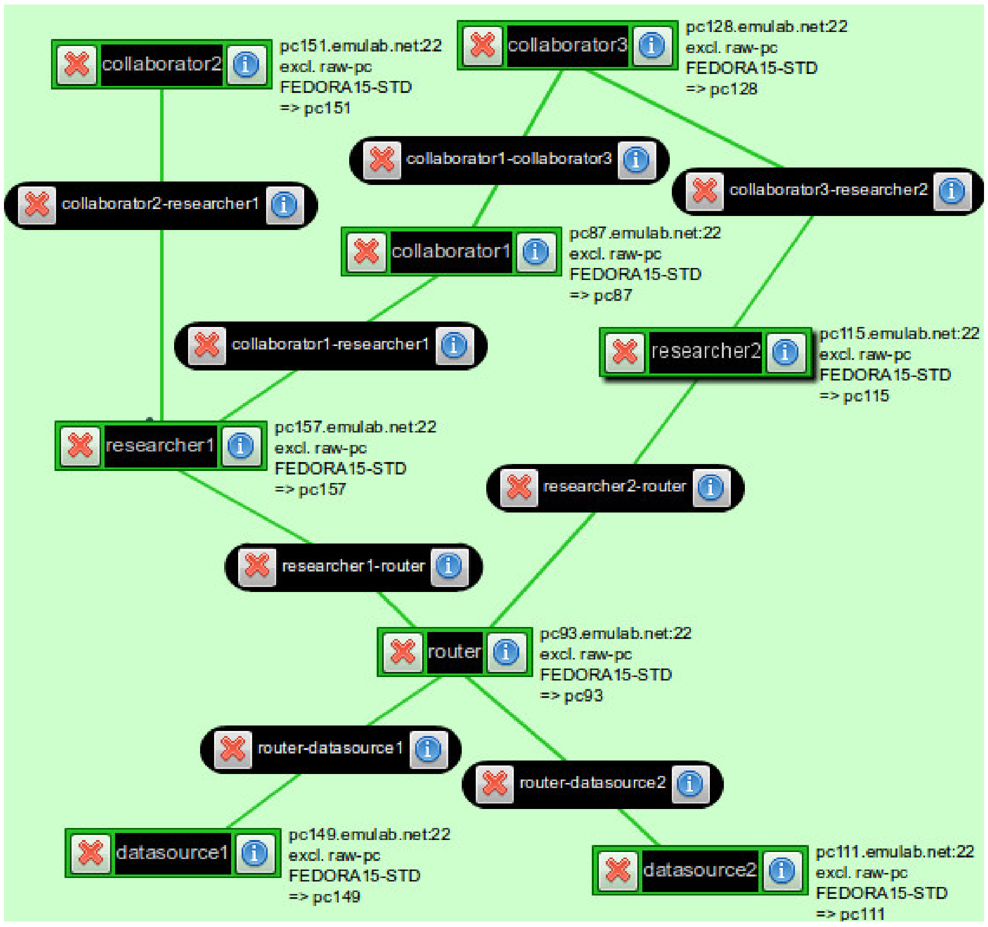

Figure 1: CCNx network for sharing precipitation data. Hostnames will change with each instantiation.

```
/opt/ccnx-atmos/client.py
```

Answer the date questions appropriately, and observe that a `.nc` NetCDF file is created in the current directory containing precipitation data for the specified dates.

Run `ccndstatus` again, and look at the statistics on the local `ccnd`. You should see both that a substantial number of content items are cached on this machine (this is the meaning of the “stored” field under “Content items”) and that many more Interests have been seen.

Next, log into `researcher2` and issue the same command. Observe that it takes much longer at each step, does not terminate, and no `.nc` file is created. This is because the `ccnd` on `researcher1` had routes to the precipitation data requested by Atmos, but `researcher2` does not. Issue the following command to install the appropriate routes:

```
ccndc add ccnx:/ndn tcp router
```

Note that the `ccndc` command takes a content path (`ccnx:/ndn`) and binds it to a network path (in this case, the host `router`, which is a valid hostname on the experiment topology, resolving to an IP address on the `router` machine). It can also be used to delete routes with the `del` subcommand.

Question 1.1 A: From `researcher2`, fetch the same data again (from 1902/01/01 to 1902/01/02), and record the fetch times reported by client.py. It prints out the time take to pull each temporary file along with the concatenation and write time. Then fetch 1902/02/03 to 1902/02/04, and record those fetch times. Fetch 1902/02/03 to 1902/02/04 a second time, and record the new times. Which transfer was longest, and which was shortest? Knowing that each `ccnd` caches data for a short period of time, can you explain this behavior?

Question 1.1 B: Browse the content caches and interests seen on various hosts in the network by loading their `ccnd` status page on TCP port 9695 in your browser (see Section 5, Hints, below). Which hosts have seen interests and have content cached, and why?

TASK 1.2: Intent Propagation

This task will demonstrate the behavior of Intent filtering during propagation.

Log into the machine `datasource1` and run the command `tail -f /tmp/atmos-server.log`. Next, run `/opt/ccnx-atmos/client.py` on both of the hosts `researcher1` and `researcher2`. Enter 1902/01/15 as both start and end date on both hosts, and press <ENTER> on both hosts as close to the same time as possible.

Question 1.2 A: What intents show up at `datasource1` for this request? (Use the timestamps in the server log to correlate requests with specific intents.) Will you observe unique or duplicate intents when both researcher1 and researcher2 fetch the same data at precisely the same time? Why?

TASK 1.3: Explore traffic patterns

For this task, you will start network monitoring services on your slice and observe the data traffic passing between hosts as you fetch various content.

Load your sliver in Flack [4] if you have not already done so by refreshing your slices (if necessary) and clicking the slice name in the left pane. Go to the Plugins tab (depicted as a puzzle piece) and click the Instrumentize button. This will take some time, as it adds and configures measurement and monitoring software to your sliver. When it completes, open the INSTOOLS portal in your web browser (Flack will prompt you) and look at the bandwidth usage between the various hosts over time in the INSTOOLS plots.

Log into the host `collaborator1` and use the Atmos client to fetch data from 1902/01/28 to 1902/02/03. Look at the INSTOOLS data usage, and observe that data is pulled from both `datasource1` and `datasource2`, through `router` to `researcher1` and ultimately to `collaborator1`. Record or remember the data rates and volumes from this transaction.

Question 1.3 A: Log into `collaborator2` and fetch the same dates. Which hosts transfer data? Why?

Question 1.3 B: Now fetch 1902/01/25 to 1902/01/31 from `researcher2`. How does this transfer differ from the previous two? Why?

3.2 Choosing Content Names

This exercise will demonstrate the importance of naming in named data networking. As the term suggests, and as you saw in Task 1.1, routing in named data networking is performed based on content names. This means that content naming has a large impact on how consumers fetch the data they require, and naming structure can heavily influence how simple and efficient data requests may be.

The Atmos package uses content names in the namespace `/ndn/colostate.edu/netsec` to serve NetCDF data. The data is sliced by time, with the suffix `pr_<year>/<month>/<day>/00` added to represent data for the specified date. Thus, clients can request precipitation data for a given day in history, or a range of days, by issuing an interest for the dates in question. The complete path for data relating to January 3, 1902 would be:

```
/ndn/colostate.edu/netsec/pr_1902/01/03/00
```

TASK 2.1: Granularity of naming

This task will require you to change granularity of content names used by the Atmos client and server for different use cases, and measure the resulting traffic changes.

Create an experimental network using the `naming.rspec` RSpec included in the assignment materials. This RSpec creates a network of the same topology as the network used in Exercise 3.1, but does not start or install the Atmos server or client. You will be modifying and installing this software yourself.

Fetch the Atmos source from `http://www.cs.purdue.edu/homes/fahmy/geni/ccnx-atmos.tar.gz`. You may fetch it directly to the testbed nodes you will be using, or to a local machine, as you prefer. You will have to copy the source code to the GENI nodes to build and run it in your experiment.

Build the Atmos source on hosts `datasource1` and `datasource2`, as well as a collaborator or researcher node of your choice by entering the source directory and running `make`. On `datasource1` and `datasource2`, you will find a NetCDF data file in /tmp, named `pr_19020101_060000.nc` and `pr_19020201_060000.nc`, respectively. On each of these hosts run the command `ccninitkeystore` and start the Atmos server with the following commands:

```
cd /tmp
$HOME/ccnx-atmos/server $DATAFILE
```

Substitute `$HOME/ccnx-atmos` for your actual Atmos build directory, and $DATAFILE for the appropriate filename as described above.

On the client host you chose, run `./client.py` from the data directory and select a few dates from January and February of 1902 for date ranges. Make note of what interests are received at the server, and how long the transfers take.

Change the client and server programs to use this URI, restricting CCNx transfers to a granularity of one month, but preserving the ability of users to retrieve at one day granularity:
`/ndn/colostate.edu/netsec/pr_<year>/<month>`

Implementation Hints: It is important understand how script.py uses NetCDF commands to extract relevant dimensions in the data files. The client (client.py) breaks up a user's request into a collection of URIs (one for each day) and fetches them sequentially. For each URI, script.py generates a list of timestamps (a floating point number indicating the number of days elapsed since January 1, 1901). From the NetCDF data files, script.py extracts the fields that lie within these timestamps. The server side changes are minimal. Once a month's granularity of data is received at the client side, it should be filtered to match the specific days requested by the user.

Retrieve a variety of dates in the January to February 1902 date range, and note again what interests are received at the server and how long the transfers take.

Question 2.1 A: What general difference do you see in network behavior between these solutions?

Question 2.1 B: Which URI scheme is more efficient in time and network resources if the user only wants a few days of data? What if the user wants a full calendar month of data?

3.3 Exploring the Impact of Caching

For this exercise you will manipulate various caching parameters for the `ccnd` daemon and observe the effects on network efficiency.

A property of named data networking is that multiple requests for the same names may return the same data. This allows intermediate nodes to cache data from one request for a given name and return it for future requests of the same name, from the same client or even a different client. When multiple clients located near each other but far from the server in the network want the same data, a node near the clients may be able to service some clients without contacting the server.

Caching behavior can be controlled by the client, the server, or the individual `ccnd` nodes between the two. Servers can optionally specify the recommended cache lifetime of data they generate, clients can optionally request that their data is fetched from the server itself (bypassing caching), and `ccnd` nodes can define their default and maximum cache lifetimes. Caches may also be affected by the available cache space and volume of interests or data items passing through the cache.

TASK 3.1: Opportunistic caching

This task will demonstrate the benefit of opportunistic caching for static data used repeatedly.

Create a GENI slice using `cache.rspec` from the handout materials. This slice has only two hosts, a data source and a data consumer. The data source serves historical precipitation data using the Atmos server, which is static information. It has a long cache timeout for this reason — five minutes. The data consumer uses a shorter cache timeout of 60 seconds, and the link between the two emulates a moderate consumer broadband link.

Question 3.1 A: From the host `consumer` run `/opt/ccnx-atmos/client.py` to fetch data from 1902/01/21 through 1902/01/24, and time the total transaction. Within 60 seconds, fetch the same data again, and time it. After 60 seconds (but before 300 seconds have passed), fetch it a third time, and time that. What is the benefit of local caching (the second fetch)? Is there perceptible benefit from server-side caching (the third fetch) when data takes some time to generate?

4 Other Projects

The following are ideas for deeper projects into named data/content-centric networking or the implementation embodied in CCNx.

- Design and implement a cache coherence or cache management protocol for content subscribers or content providers to communicate with intermediate nodes to improve caching strategies in the presence of data with dissimilar caching requirements.

- Develop an application-specific caching mechanism for some data type (perhaps NetCDF) and deploy it in CCNx.
- Design and implement a protocol for `ccnd` to communicate directly with other `ccnd` instances via raw Ethernet frames without IP, ARP, or the rest of the TCP/IP stack.

5 Hints

These hints and answers to common questions may help you debug problems you encounter while performing the exercises in this assignment.

- Remember that you must run `ccninitkeystore` before invoking any commands that use `ccnd`. If CCNx commands don't seem to be "doing anything", make sure you have the file `.ccnx/.ccnx_keystore` in your home directory. If it is not present, you have forgotten to initialize the keystore!
- If you are told that CCNx-related commands such as `ccndc` are not found, make sure that `/opt/ccnx-0.6.1/bin` is in your shell's path.
- The status of `ccnd` can be queried either by logging into a host and running `ccndstatus`, or by loading the web page served by `ccnd` on port 9695. For example, if your `researcher2` node is on `pc308.emulab.net` and you want to query its forwarding table, you can point your web browser at `http://pc308.emulab.net:9695/`.
- If the INSTOOLS portal shows a blank page, and you are running the Chrome browser, click the shield icon at the right of the URL bar and select "Load Anyway".

References

[1] T. Berners-Lee, R. T. Fielding, and L. Masinter. Uniform resource identifier (URI): Generic syntax. RFC 3986. Available at `http://www.ietf.org/rfc/rfc3986.txt`, Jan. 2005.

[2] CCNx URI Scheme. http://www.ccnx.org/releases/latest/doc/technical/URI.html.

[3] NetCDF. http://www.unidata.ucar.edu/software/netcdf/.

[4] M. Strum. Flack manual. http://www.protogeni.net/trac/protogeni/wiki/FlackManual.

[5] The CCNx Project. http://www.ccnx.org/.

[6] The Geni Project Office Wiki. Omni. http://trac.gpolab.bbn.com/gcf/wiki/Omni.

[7] The GENI wiki. http://groups.geni.net/geni.

[8] The Network Security Group at Colorado State University. http://www.netsec.colostate.edu/.

[9] The ProtoGENI Project Wiki. ProtoGENI tutorial. http://www.protogeni.net/trac/protogeni/wiki/Tutorial.