AsiaFI NDN Hands-on Workshop 2012

Homework

Overview and Goals

This homework is designed to help you get hands-on experience with the CCNx OpenSource platform and introduce higher level Named Data Networking topics like name design and interest construction. This homework will expose you to distributed system design and debugging. After completing this homework you should be able to apply the concepts to your own projects and ideas.

There are many ways to implement the homework. Different approaches can have the same outcome but achieve it in different ways with different tradeoffs. The implications of design are important to experience and understand so they can be applied in your projects and deployments.

This homework has two core components: a publisher and a consumer. The publisher component periodically announces its existence in an ‘active users’ namespace and immediately announces any messages you type in a ‘participant messages’ namespace. The consumer component expresses interests in the ‘active users’ namespace to discover publishers then expresses interests in each publisher’s ‘messages’ namespace to get and display that publisher’s messages.

Description

In this homework you will be building a simple, distributed, text-based short message service. You can think of it as a broadcast-based SMS or light-weight Twitter. At a high level, each user (workshop participant) will create a user identity and post messages identified by the unique identity and a sequence number. In the following paragraphs, we note naming conventions and explicitly define the namespaces to permit interoperability between participants.

As discussed during the first day of the workshop, the need for name components that represent identity, versioning (temporal evolution) and/or segmentation are common to many applications of NDN and there are existing library routines and conventions for each:
- `<uid>` is the name component with a reserved prefix of %C1.M.K followed by the digest of a public key.
- `<version>` is the name component with a reserved prefix of %FD followed by bytes that are usually based on a timestamp.
- `<segment>` is the name component with a reserved prefix of %00.
User namespace:

<uid> is the unique ID of the participant – the digest of the user’s public key. For user Content Object names, <version> can be a timestamp or any strictly increasing number (but the number should not go backwards is the program is stopped and restarted). The segment number should be created by the underlying library mechanisms. The User namespace template is:

\[ \text{ccnx:/asiafi/2012/ndnworkshop/homework/users/}\langle\text{uid}\rangle/\langle\text{version}\rangle/\langle\text{segment}\rangle \]

Participant message namespace:

For message names, the <uid> is the same as used above to announce participation. The <seqnum> is the monotonically increasing sequence number and should be an integer incremented by one for each message, starting with 0. The <segment> component is again created by the underlying library mechanisms. The message namespace template is:

\[ \text{ccnx:/asiafi/2012/ndnworkshop/homework/messages/}\langle\text{uid}\rangle/\langle\text{seqnum}\rangle/\langle\text{segment}\rangle \]

Your application should create your user identity using library functions. Next, you should take text input and output it as message objects in the your identity’s message namespace. In another window, you can simply stream all messages from each unique id. For example, you might have output that looks like:

uid1: Joining workshop homework!
uid1: Hi everyone! I have something running!
uid1: Is anyone there?
uid2: I am joining the workshop homework.
uid1: Hi uid2! Welcome!
uid2: Thank you!
uid2: uid1, I just got your other messages. Great job with the homework!
uid3: Participating in homework exercise.
uid2: Hi there, uid3!
uid3: Hi uid2! How are you
...

To get started, you need your program to have two functional components. First, you need to have a component to create your identity and periodically publish an object to “announce” your participation. This code will create Content Objects in the user namespace. Second, you will need to create a component that will discover new users in the namespace. As connectivity changes and participants implement the homework, new users will appear. Your implementation must continuously discover new participants as they become visible to you.

You also need to create and retrieve content. When you create your user id and publish a content object into the user namespace, you will also want to start creating content to
publish in the message namespace. You might want to start with an announcement that you have joined the workshop service then a read-and-send loop that sends each line input as a new message. These messages should have an integer denoting the sequence number of the messages as the version number (described above). Using this convention allows your message gathering code to detect and fill holes. In addition to detecting lost messages, your code will need to handle cases where some messages are not available. You want to avoid having your code block on a message that cannot be retrieved.

Your application should produce UTF-8 encoded text strings as the data of the Content Objects in the message namespace.

For this assignment, a simple text interface is encouraged. Since how you display the data does not impact the functionality, feel free to simply stream the messages and any debugging information that you wish.
Extensions

1. Fill in holes from missed messages. For example, if you first receive a message with sequence number component 5, our shared naming convention conveys the knowledge that messages 0 through 4 also exist and you will need to construct interests to retrieve them. You can simply stream the messages as you discover them.

2. Display messages in order. As you discover new participants and have topology changes, you will encounter holes in the message namespace. In this extension, you will display messages from each user in the order they were created. Since we do not have synchronized clocks, it is not trivial to get a total ordering across all users, but it is possible to have the messages for each individual participant to be in the correct order (as permitted by connectivity).

3. Users may post messages in response to previously posted messages. You can publish these replies with links to the previously published messages. In this extension, you can display the messages as replies by following the link in the content object. To allow interoperability, please use the following convention in the content object (UTF-8 encoded strings). Put the your message (reply) first then follow this with two new lines (“\n\n”) and the URI encoding of the full CCNx Content Name (including digest) at the end of the messages. This method will allow for nested messages and you should be prepared to deal with nested Links.

Example for message 5 linking to message 4:

<table>
<thead>
<tr>
<th>CCNx Content Object Name:</th>
<th>ccnx:/asiafi/2012/ndnworkshop/homework/messages/&lt;uid&gt;/5/%00</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCNx Content Object Body:</td>
<td>Announcing that I have implemented links!</td>
</tr>
<tr>
<td></td>
<td>ccnx:/asiafi/2012/ndnworkshop/homework/messages/&lt;uid&gt;/4/%00/&lt;digest&gt;</td>
</tr>
</tbody>
</table>

4. One new aspect of NDN is to allow information consumers to verify that the content producer is a trusted entity. In this extension, you can verify that the publisher of the content object matches the unique id, this establishes a level of trust for the content. Not only will the content be verified against transmission errors, but it will also be validated so impostors can be detected.

5. You may want to be able to view or retrieve messages when isolated from other users. While some content could be cached in ccnd, this is not a dependable source of information. To mitigate this, you can save messages to a repository to make them available even when partitioned from other users (or the network). You can use library functionality to achieve this, please see sample code on how to save
objects to a repository. (NOTE: This will also require that you run a repository (ccnr) on your machine in addition to ccnd.)

6. While using a hash of a user’s key does uniquely identify them, it is not the traditional method for addressing a person. In this extension, you will allow users to securely identify themselves with a “friendly” name. To allow interoperability, please use the user namespace object as the holder for the friendly name. For example:

```
ccnx:/asiafi/2012/ndnworkshop/homework/users/<uid>/<version>/<segment>
```

would contain the UTF-8 encoded string:

```
Bob Smith
```

As the friendly name is updated, new versions of the user namespace object should be created. The friendly name can be used when displaying the messages for each user, and can be tested by attempting to assign a friendly name to a different user. If your code is correct, this attempt should be discovered. You can either choose to display the attempt as a malicious activity (recommended) or drop the message.

7. As previously discussed, without synchronized clocks we do not have a global ordering for messages. In this extension, use a technique such as Vector Clocks to create a total ordering. Other techniques can be used to impose global ordering as well.