NDN File Transfer Application with Congestion Control
Shuo Yang

1 Development Environment Setup

1.1 Code repository

`git@dyadis.cs.arizona.edu:ndn_cc_2016`

Repository structure:
`README.md`: step by step instructions on how to setup the development environment
`apps/file-transfer/`: file transfer application
`docs/`: documentation
`scripts/`: scripts for installing virtual machine, running minindn experiments, etc.
`results/`: experiment results

1.2 Development environment setup

The development environment I use is a Ubuntu 15.04 VM with NDN related software (ndn-cxx, nfd and minindn) installed.

Please follow the steps in the `README.md` to set up the development environment and how to compile the code and run it.

One thing we need to be aware of about minindn is that you must put the experiment scripts into minindn's source code structure under `mini-ndn/ndn/experiments` in order to run the experiments. And every time we make changes to the scripts, we must reinstall minindn to make them effective.

2 Implementation

2.1 Queue size calculation

First, we need to figure out how big a data packet is. For a ndn data packet with content payload with the size of 1024 bytes, `tcpdump` shows that the following:

```
07:47:39.952381 In 26:44:97:59:e6:cf (oui Unknown) ethertype IPv4 (0x0800), length 1449: (tos 0x0, ttl 64, id
6388, offset 0, flags [none], proto UDP (17), length 1433)
1.0.0.2.6363 > 1.0.0.1.6363: [bad udp cksum 0x0799 − > 0xbb08!] UDP, length 1405
```

Based on this, we can infer the packet structure as follows:
- total data packet size is 1449 bytes
- UDP packet size: 1433 bytes
- UDP payload (ndn data packet) size: 1405
- ethernet header: 1449 - 1433 = 16 bytes
- IP header + UDP header: 1433 - 1405 = 28 bytes
- ndn header overhead (signature, etc): 1405 - 1024 = 381 bytes

Next, I measured the RTT for a single pair of interest and data packet (with content payload: 1024bytes) traveling through the linear topology. I repeated the same process 5 times and got the average.
Below is the RTTs I got:
71ms
75ms
72.8ms

71.9ms
75.2ms

---

avg: 73.2ms

So the queue size is calculated as:

$$\begin{aligned} queue\_size &= (rtt * bandwidth)/data\_packet\_size \\ &= (73.2ms * 5Mbps)/1499bytes \\ &= 31.6\#\text{of data packets} \\ &\approx 32 \end{aligned}$$

2.2 AIMD implementation

1. When a data packet was received, if $cwnd < ssthresh$, increment $cwnd$ by 1 (additive increase, slow start), otherwise increment $cwnd$ by $1/cwnd$ (linear increase, congestion avoidance).
2. When an interest packet was timed out, set $ssthresh$ to half of $cwnd$ (multiplicative decrease) and reset $cwnd$ to 1.

2.3 Sequence hole detection implementation

This is implemented as an optional congestion control scheme and can be turned on or off by a command line option. By default, it is turned off.

The assumption for this scheme is that most time data packets should arrive in order.

1. When a data packet was received, if it arrived in order (in the same order as the corresponding interest packet), then we use AIMD scheme as mentioned above to adjust the size of congestion window.
2. If it arrived out of order, we keep a counter on how many out-of-order data packets have received.
3. If the counter is below a certain threshold (5 by default), we don't adjust the size of congestion window.
4. If the counter is above the threshold (5 by default), we set $ssthresh$ to half of $cwnd$ and set $cwnd$ to $ssthresh$ (fast recovery).

2.4 Timer and timeout implementation

1. For each sent interest, we keep track of the time it was sent and RTO value calculated at the time it was sent.
2. Every 10 ms, check RTO timers for each sent interest, if the timer expires, adjust the congestion window size according to the AIMD scheme and retransmit the timed out packet.

3 Experiment

3.1 Experiment setup

Topology

We are currently using a linear topology for debugging purpose:
consumer——router1——router2——producer
host-router links (10Mbps, delay=10ms), router-router link (5Mbps, delay=10ms, max_queue_size=32)

File transfer applications

We ran four different file transfer applications to download a file of size 10MB, and each was repeated 5 times.

1. File transfer application with a fixed congestion window size.
2. File transfer application with AIMD scheme.
3. File transfer application with AIMD+Hole_detection scheme.
4. FTP based on TCP/IP.

3.2 Parameters

```
MaxRTO = 10s; // maximum RTO value
alpha = 1/8; // filter gain
beta = 1/4; // filter gain
RTT variance multiplier = 4
RTO backoff multiplier = 2
initial cwnd = 1
initial ssthresh = 200
additive increase step = 1
multiplicative decrease factor = 0.5
out-of-order packets counter threshold = 5
retransmission timer check interval = 10ms
```

3.3 Metrics

1. *Download time*: total time (in seconds) it takes to download the file.
2. *Actual Throughput*: calculated as (total number of data packets received * data packet size) / (total download time).
3. *Effective Throughput*: calculated as (number of data packets received * data packet size) / (total download time).
4. *Loss rate*: calculated as (total number of retransmitted packets) / (total number of packets received).
5. *congestion window size*: measured every 10ms

4 Results

NDN file transfer application with fixed cwnd=50

	download time (s)	actual throughput (kbps)	effective throughput (kbps)	loss rate
	24.1	4932.18	4932.18	0.2%
	23.9	4964.96	4964.96	0.2%
	23.8	4981.83	4981.83	0.2%
	23.8	4980.4	4980.4	0.2%
	23.8	4987.08	4987.08	0.2%

NDN file transfer application with fixed cwnd=100

	download time (s)	actual throughput (kbps)	effective throughput (kbps)	loss rate
	25.1	4720.92	4720.92	5.6%
	25.5	4654.68	4654.68	5.7%
	25.6	4634.07	4634.07	5.6%
	26.8	4426.16	4426.16	5.6%
	24.6	4818.9	4818.9	5.6%

NDN file transfer application with fixed cwnd=200

	download time (s)	actual throughput (kbps)	effective throughput (kbps)	loss rate
	25.6	4640.69	4640.69	17.2%
	26.2	4520.9	4520.9	17.5%
	26.2	4520.48	4520.48	17.5%
	26.3	4514.49	4514.49	17.6%
	24.7	4811.88	4811.88	17.1%

NDN file transfer application with AIMD scheme

	download time (s)	actual throughput (kbps)	effective throughput (kbps)	loss rate
	50	4290.2	2373.14	100%
	46.5	4520.15	2551.64	100%
	54.9	3898.3	2156.63	100%
	49.4	4236.13	2401.5	100%
	47.9	4472.26	2475.99	100%

NDN file transfer application with AIMD+Hole scheme

	download time (s)	actual throughput (kbps)	effective throughput (kbps)	loss rate
	42.9	5504.04	2767.84	100%
	42.8	5779.22	2773.88	100%
	44.6	5401.22	2659.55	100%
	46.2	5205.68	2569.45	100%
	46.7	4937.29	2541.65	100%

FTP based on TCP/IP

	download time (s)	throughput (kbps)
	17.58	4653.6
	17.58	4654.4
	17.59	4656.8
	17.58	4654.7
	17.59	4656.4