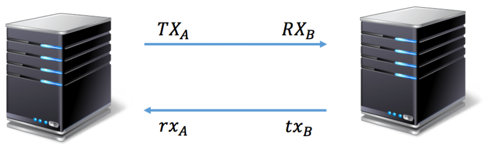
EE392K Problem Set 3

Please email your solution to
zyin@stanford.edu by Fri, Mar 3, 2017

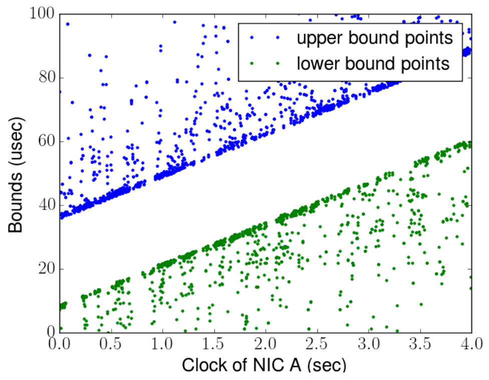
February 15, 2017

1 Clock Synchronization

Recall that clock synchronization uses the following 4 time stamps: TX_A, RX_B, tx_B and rx_A.



(a) Timestamps between Probing Pair



(b) Upper and Lower bounds

Let t_A and t_B be the times at servers A and B when the true time is t. Let $t_B = t_A + \Delta$. The 4 timestamps provide upper and lower bounds on Δ:

$$tx_B - rx_A < \Delta < RX_B - TX_A. \tag{1}$$

Remember for a pair of servers A and B who are probing each other, with the time stamps, you can plot upper and lower bounds on Δ, which looks like figure 1b.

The slope and intercept of the middle separating line are the drift and offset between A and B. In the last problem set, we explored three different ways to determine the middle line, namely SVM, neural network for fitting the line, and neural network for fitting points along the line. In this problem, we explore two more techniques, namely the coded packets and network effect, both of which help to obtain more accurate clock synchronization.

Coded Probes

Probes can experience various noise and delays in a network, including NIC timestamping noise, queuing delay and switching delay. The adversarial effects by the noise and delays can be mitigated by identifying the probes that experience minimal amount of delay and noise. Coded probes is a technique that harnesses a simple yet effective idea: if two closely-separated packets come out of a network with the same separation, it is likely that both experienced no faulty timestamps or queuing delays. Figure 2 is an illustration of a pair of probes with a small time separation s. A larger separation means packet 2 is delayed more while a smaller separation indicates packet 1 is delayed more.

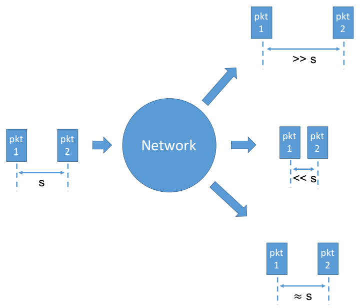


Figure 2: Illustration of Coded Probes

A guardband μ is used to decide whether a pair of coded packets are accepted. Suppose the separation at the transmitter is s, and the separation at the receiver is s'. This pair will be rejected if $|s' - s| > \mu$.

Exercise 1.1

Use the simulation program **generateTimestamp.py** to simulate the timestamps in a 128-server, 3 stage, 40 Gbps Ethernet test environment.

1. Directly use SVM for middle line estimation from last problem set. The slope and intercept are the drift and offset between the two clocks. Compare the offsets and drifts reported by SVM with the ground truth offsets and drifts for all server pairs probing each other. Report the mean and standard deviation of the estimation error, for both drift and offset.

2. Apply coded packet filter with a 50-nanosecond guardband. After that, use SVM on the remaining packets to obtain the middle line. Report the mean and standard deviation of the estimation error, for both drift and offset. You should observe an increase in accuracy, compared with the previous exercise.

Network Effect: Better to Sync More than a Few

As pointed out by Einstein, clock synchronization should be reflexive (A is synched with A), symmetric (if A is synched with B, then B must be synched with A), and transitive (if A is synched with B and B is synched with C, then A must be synched with C). In particular, the transitivity can be used to discover inconsistencies in clock synchronization and correct them. The example in Figure 3 illustrates the point. In figure 3, after mutual synchronization, clocks A and B believe that the offset between their clocks ($t_B - t_A$) is 20 units of time, indicated in the figure with directed edges ($A \overset{20}{\rightarrow} B$). However, the truth (shown parenthetically in green) is that B is ahead of A by only 10 units of time. But, A and B can never discover this error by themselves. In (b), a third clock C has undergone pairwise synchronization with A and B, and the resulting pairwise offsets are shown on the directed edges. Going around the loop $A \rightarrow B \rightarrow C \rightarrow A$, we see that there is a loop offset surplus of 10 units! This immediately tells all three clocks there are errors in the pairwise synchronization. The bottom of Figure 3 (b) shows two possible corrections to the pairwise estimates to remove the loop surplus.

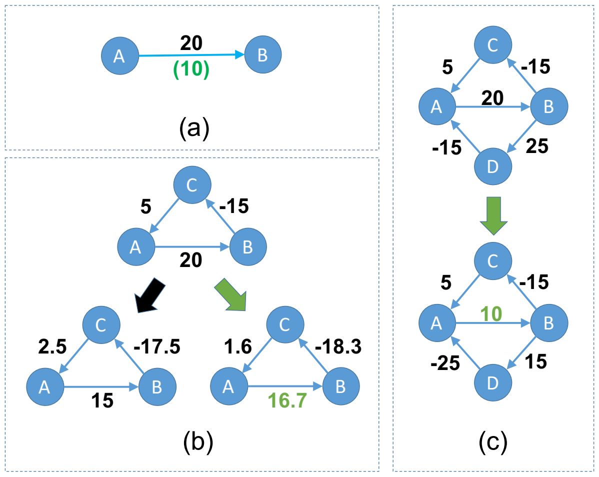


Figure 3: Illustration of Loop Correction

Loop Composition Matrix

Suppose there are L loops in a cycle basis (click HERE for more info on cycle basis), for the probing graph $G = (V, E)$. Each loop consists of a set of directed edges. If we sum up the discrepancies along a loop, and end up with a number different from 0, it is an immediate indication of error in the estimation of clock differences.

We construct a *Loop Composition Matrix* A, where each column of A corresponds to an edge in the probing graph, and each row in A corresponds to a cycle in the cycle basis. Specifically, $A_{ij} = 0$ if edge j is not present at cycle i, $A_{ij} = 1$ if cycle i contains edge j, and $A_{ij} = -1$ if cycle i contains edge j with the reversed direction. Suppose the vector of ground truth discrepancies between clocks sitting on two ends of the edges are Δ, we will have the following identity:

$$A\Delta = 0$$

For the estimated discrepancy vector Δ^P from SVMs, we first multiply it with the pseudo inverse of A, $A^\dagger = A^T(AA^T)^{-1}A$, to obtain an estimate of the estimation error. Then we remove the error from Δ^P to get a finalized estimate Δ^F, which formally equals

$$\Delta^F = (I - A^\dagger)\Delta^P$$

Under the assumption that the estimation errors are IID, it can be shown that this procedure is capable of removing $\sqrt{\frac{|V|-1}{|E|}}$ of the noise, which roughly equals 68.4% when each server probes 10 other servers.

Exercise 1.2

1. Construct the loop matrix A by completing the starter code. You may want to use python modules like **networkx** for basic graph theory operations.

2. Write the rest of the loop correction algorithm. Test it on the SVM estimates obtained in exercise 1.1. Compare the corrected discrepancy vector $\|\Delta^F - \Delta\|_2$ with $\|\Delta^P - \Delta\|_2$, and report the reduction of error.

2 Network Reconstruction

The timestamps of probes can help us synchronize clocks of different servers, but they can do more. The one-way delay of a probe is defined as its RX timestamp minus its TX timestamp. The one-way delay consists of propagation, switching and queueing delay. The propagation and switching delays, compared with queuing delay, are less significant for the following reasons. As electric signals transmits at a speed of $0.7c$ on a **Cat5** cable, the propagation delay is roughly $4.7ns/m$, with negligible variations. At the same time, typical switching delay for a CISCO switch is $4\mu s$, and the variation is around 100s of nanoseconds. As a result, a large one-way delay, usually on the order of hundreds of microseconds to milliseconds, indicates the packet encountered a large queue somewhere on its path. With this simple observation, we can use probe timestamps to reconstruct the size of the queues, at a particular time interval.

NS-3 Simulator

NS-3 is a network simulation tool that's widely used for research. In this exercise, you are going to simulate a 64 server, 3-stage 1GbE network. Timestamps of probes will be recorded for queue reconstruction. The ground truth queue sizes will also be reported by NS-3, which can be used to compare with the reconstruction.

Setting up NS-3

You can install NS-3 either on your own Linux computer, or on corn. If you use your own computer, make sure all prerequisites are met according to https://www.nsnam.org/wiki/Installation. For corn, the prerequisites are satisfied.

Then download the edition of ”ns-3-dev”:

```
 mkdir repos
 cd repos
 hg clone http://code.nsnam.org/ns-3-allinone
 cd ns-3-allinone
 ./download.py -n ns-3-dev
```

After downloading, get into the folder ns-3-dev

```
hg checkout 043544eef3ed
copy "ecmp.patch" to the folder ns-3-dev
hg import ecmp.patch
```

Then build it: Go to the folder ns-3-dev, run:

```
./waf configure --enable-examples --enable-tests --disable-gtk --disable-python
```

DO NOT add –enable-sudo. Then run:

```
./waf
```

to do compiling. Finally run the tests:

```
./test.py
```

Done! For running simulations, use the provided .sh file. You can change the commands and parameters to run simulations with different time, logging, etc.

```
============================================================
```

Trouble shooting: During compiling, if you see error messages like:

```
../src/network/utils/ipv6-address.cc: In function ’uint32_t ns3::lookuphash(unsigned c
../src/network/utils/ipv6-address.cc:64:26: error: typedef ’ub1’ locally defined but n
   typedef unsigned  char ub1;   /* unsigned 1-byte quantities */
                          ^
cc1plus: all warnings being treated as errors
```

The following procedure should resolve this issue. Open ns-3-dev/src/network/utils/ipv6-address.cc, then delete or comment line 64 and save. Now run ./waf, and error should be fixed.

Exercise 2.1: Run NS-3 Simulation

After installing NS-3 on your computer, run the experiment with the .sh script. The script also configures the parameters of the simulation, including

- Reconstruction Interval. This parameter determines the separation between two consecutive probes.
- Length of simulation. A 0.5-second run would suffice, as it contains fifty 10-ms intervals, at which granularity will we do the reconstruction.
- Number of runs. In this exercise, simulating one 0.5-second run would suffice. With more runs, you could get more data but it also takes longer time.
- K, the number of destinations to be probed by a server. Bigger K gives better coverage, but it at the same time takes more bandwidth, computation and time to simulate. Choosing K between 10 to 20 will strike a balance between coverage and efficiency.

Before running your own scripts, pre-process the data by running two scripts provided, **data-sim-queuelength.py** and **data-getRTT.py**. Parameters are **4 4 4 4 4**.

Queue Reconstruction

Since

$$\textit{Oneway delay} = \textit{queuing delay} \ + \ \textit{propagation delay},$$

once the one-way delay of a probe is obtained, we can subtract the propagation delay, and obtain the queuing delay of the probe.

Exercise 2.2: SVD of Delay Matrix

For each reconstruction interval, construct the delay vector of all probe packets sent within that interval. Stacking the delay vectors, obtain a matrix $M \in \mathbb{R}^{t\times d}$. Compute the singular values of $M^T M$, and plot its singular values.

1. By observing the plot, can you tell the number of stages of the network?
2. Can you tell the number of independent paths?

Exercise 2.3: LASSO Reconstruction

In the starter code, we provided helper functions that read the outputs from NS-3 simulation. In particular, we load 1) the RTTs of the probes, 2) the queues each probe went through and 3) the ground-truth queue size. We will use 1) and 2) to do the reconstruction, and verify using 3).

1. For each reconstruction interval, construct the probe-queue incidence matrix A, and the delay vector D. Hint: each column of A corresponds to a queue, and each row of A corresponds to a probe. $A_{ij} = 1$ if and only if probe i went through queue j.

2. Use LASSO to solve for Q, the queue size vector. In particular, find

$$\arg_Q \min \|AQ - D\|_2^2 + \alpha \|Q\|_1$$

where α is chosen to be

$$16 \times 10^{-6} / \|Q\|_0$$

3. Choose three queues in Q, and plot their reconstructed sizes against the ground truth across all reconstruction intervals.

4. Report the relative root mean-square error (RMSE), which is defined as

$$\frac{\|\hat{Q} - Q\|_2}{\|Q\|_2}$$

Neural Network Reconstruction

Now let's switch to a neural network for queue reconstruction. As discussed earlier, neural networks have the following advantaged:

- It does not depend on the underlying model. For LASSO, the A matrix needs to be derived from the network topology. For neural network, it only needs to observe enough training data, and the relationship between input and output will be automatically learnt.

- When making predictions, neural networks are fast. Solving for LASSO needs expensive computations, whose complexity grows super-linearly. For the 3 stage, 64 server network, LASSO takes 48ms to reconstruct a 1ms interval, which makes real-time inference impossible.

Exercise 2.4: Neural Network Reconstruction

1. Implement a neural network for reconstruction by completing the rest of the starter code. The input of the neural network is the delay vector D and the output is the queue size vector Q. For the loss function, use square loss.

2. Train the neural network using simulated data from NS-3.

3. Choose three queues in Q, and plot their reconstructed sizes against the ground truth across all reconstruction intervals.

4. Report the relative root mean-square error (RMSE). You should observe similar or better performance than LASSO.

Hierarchical Reconstruction

Instead of running LASSO to solve for all queues at the same time, we consider an alternative scheme. First, we consider the “level 0” queues, those from ToRs to the servers. Using same-rack probing, we get probes that only go through those queues. As a result, we can reconstruct these queues from only the same-rack probes.

Now we go one level above, and consider “level 1” queues, those between ToRs and L2 switches. Using same L2-block probing, we get probes that only go through the “level 0” and “level 1” queues. Moreover, since we already reconstructed “level 0” queues, we can simply subtract them. As a result, we end up with delays associated only with the “level 1” queues. Again, LASSO can be used to reconstruct these queues. By repeating this procedure, we can reconstruct higher levels queues. Figure 4 illustrates the levels of queues to be reconstructed.

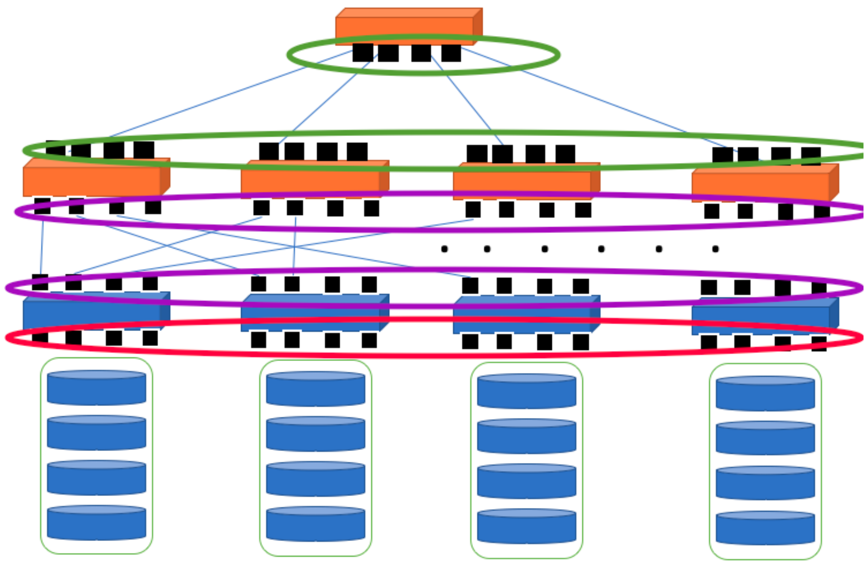


Figure 4: Hierarchical reconstruction in a three-stage network. Red: level 0 queues; Purple: level 1 queues; Green: level 2 queues

Exercise 2.5: Hierarchical Reconstruction

1. Modify **rpc_probe_new.cc** in folder **scratch** to enable same-rack and same L2-block probing, and re-run the NS-3 simulation. Make sure all the queues are covered by at least one probe.

2. Implement the hierarchical reconstruction algorithm. Report the relative RMSE at each level. You can consider only queues whose sizes are larger than 5kB.

3. Profile the running time of the hierarchical reconstruction, and compare it with full one-shot LASSO. How much reduction do you observe?