Defeat MITM Attack with Reinforcement Learning (Supplement Material)

Yang Hu yh2948
Xiaotian Hu xh2332

1. Appendix.A Likelihood Ratio Derivation

For a time span $(t - t_0, t)$, one of nodes in the network is kept being detected by the probe and generate a result sequence:$[d(t - t_0), d(t - t_0 + 1), \dots, d(t)]$ where $d(i) = 1$ or 0. In the sequence, 1 denotes the nodes is detected to be intruded and 0 denotes the nodes is not intruded. The accuracy the probe is denoted as ACC, which means the possibility that probe return 1 when the nodes is actually intruded. And the false omission rate is denoted as FOR, which means the possibility that the probe return 1 when the node is acutually free from intrusion.

Then the likelihood that the node is intruded is:

$$lr1(t) = \prod_{i=t-t_0}^{t} P(d(i)|1) \tag{1}$$

where

$$P(d(i)|1) = \begin{cases} ACC, & d(i) = 1 \\ 1 - ACC, & d(i) = 0 \end{cases} \tag{2}$$

While the likelihood that the node is not intruded is:

$$lr0(t) = \prod_{i=t-t_0}^{t} P(d(i)|0) \tag{3}$$

where

$$P(d(i)|0) = \begin{cases} FOR, & d(i) = 1 \\ 1 - FOR, & d(i) = 0 \end{cases} \tag{4}$$

To evaluate the probability that the node is intruded, we take the ratio of likelihood of intrusion and non-intrusion:

$$\begin{aligned} lr(t) &= lr1(t)/lr0(t) \\ &= \prod_{i=t-t_0}^{t} \frac{P(d(i)|1)}{P(d(i)|0)} \\ &= \prod_{i=t-t_0}^{t} l(i) \end{aligned} \tag{5}$$

where

$$l(i) = \begin{cases} \frac{ACC}{FOR}, & d(i) = 1 \\ \frac{1-ACC}{1-FOR}, & d(i) = 0 \end{cases} \tag{6}$$

Figure 1, 2, 3 show the likelihood ratio swinging during the testing. The likelihood ratio keeps changing because of the changes of attack pattern.

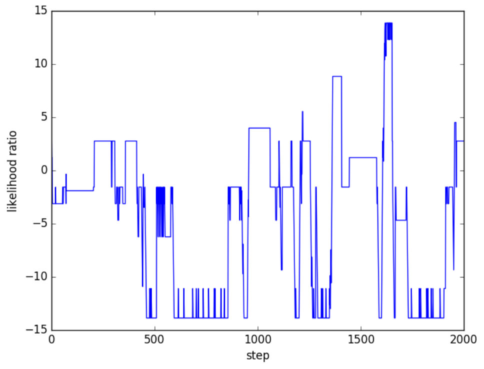

Figure 1. likelihood ratio swinging of node4

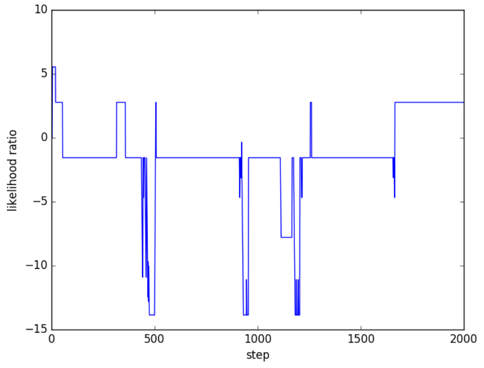

Figure 2. likelihood ratio swinging of node5

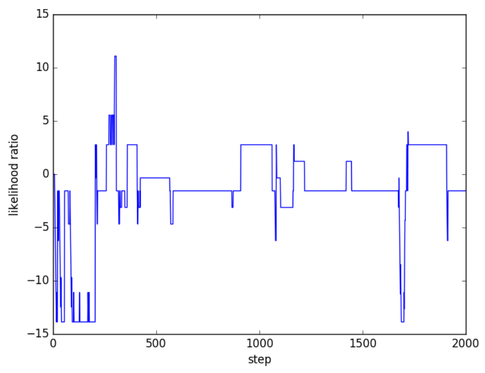

Figure 3. likelihood ratio swinging of node10

2. Appendix.B Persudo Code of Neural Q-Learning

Algorithm 1 Asynchronous NAF - N collector threads and 1 trainer thread

```
Initialize $D$, $MemSize$, $Step$
Initialize $\omega$
Initialize $Exp$
Initialize $BatchLen$
for step $= 1, MemSize$ do
    action $a_t = \epsilon -$ greedy$(actionlist)$
    execute $a_t$ and observe state $s_{t+1}$, reward $r_t$
    append $(s_t, a_t, r_t, s_{t+1})$ to $D$
    $s_t = s_{t+1}$
    if step $> EXP$ then
        randomly sample a batch of transitions
        $(s_i, a_i, r_i, s_{i+1})$ from $D$
        $y_i = r_i + \gamma \max'_a Q(s_{i+1}, a'|\omega)$
        train Q value approximator with $y_i, s_i$ pair batch
    end if
end for
for t=1,T do
    Execute $\boldsymbol{u}_t$ and observe $r_t$ and $\boldsymbol{x}_{t+1}$
end for
```

Where D is the replay memory, $MenSize$ is the max memory size, $Step$ is the number of steps for training, Exp is the least replay memory size to start training, $BatchLen$ is the batch size for each training and ω is the newtork weights

In our experiments, we set training step $M = 30000$. We build a 5-layer feedforward network to serve as the Q-value approximator, the size of each layer is: 135,270,108,52,8. Gredient descent is used to train the network.

3. Appendix.C Optional Path Generation

In our experiment, we build a simple path finding algorithm to mimic the routing mechanism. The algorithm will find the simple paths between the given start and end nodes and remove the paths which form loops, it is depicted in the figure 4 and figure 5below. In this illustration, the green edges form a path between the start and end node.

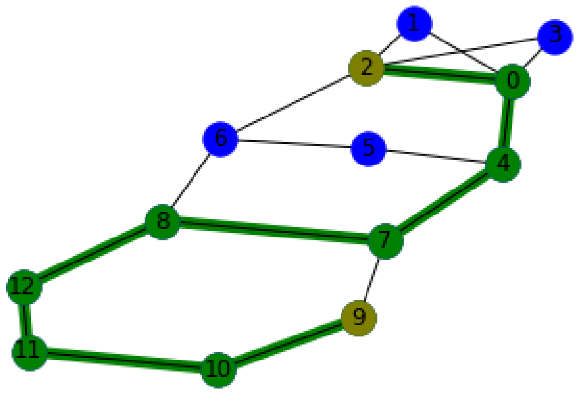

Figure 4. One of the path found: 2,0,4,7,9. The nodes in this path do not form any loop and, therefore, is regarded as one of the optional path

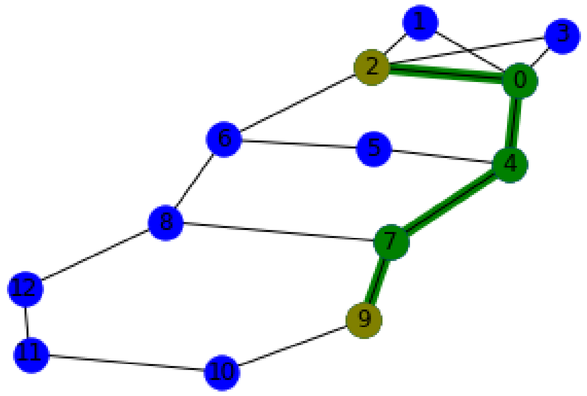

Figure 5. One of the path found: 2,0,4,7,8,12,11,10,9. A loop: 7,8,12,11,10,9,7 exist in this path, hence it is discarded

we choose 4 four paths among those found path as the optional routing. The number of optional routs is 4 because it is found that for almost all the start-end pairs, the number of found paths is less than 4.