OLAF: Operation-Level Traffic Analyzer Framework for Smart Grid

Wenyu Ren*, Steve Granda*, Tim Yardley*, King-Shan Lui† and Klara Nahrstedt*
* University of Illinois Urbana-Champaign, Urbana, Illinois, USA
Email: {wren3, sgrand2, yardley, klara}@illinois.edu
† The University of Hong Kong, Hong Kong, China
Email: kslui@eee.hku.hk

***Abstract*—The current Smart Grid supervisory control and data acquisition (SCADA) systems are primarily protected at the perimeter level with firewalls at the boundary of the networks. However, besides the attacks coming from the external Internet, internal attacks are equally concerning. Therefore, systems need to be protected from internal attacks within the perimeter. In Smart Grid, the Field Devices (FDs) are resource-constrained devices that do not have the ability to provide security analysis and protection by themselves. And the commonly used industrial control system protocols offer little security guarantee. To guarantee security inside the system, analysis and inspection of both internal network traffic and device status need to be placed close to FDs to provide timely information to power grid operators. For that, we have designed a unique, extensible and efficient operation-level traffic analyzer framework named OLAF. The time overhead and performance evaluations of the analyzer confirm efficiency and accuracy under our simulated Smart Grid operational traffic.**

I. Introduction

For the Smart Grid supervisory control and data acquisition (SCADA) systems nowadays, various security mechanisms (e.g., firewalls and gateways) are applied to the boundary of the infrastructure to inspect and secure the information exchanged with external entities. However, data within Smart Grid networks, gathered by the internal field measurement devices, is usually not visible to the operators and not secured at the same security levels as communication with external entities. As it is shown in Figure 1, Smart Grid wide-area networks connect Smart Grid substation field networks with operational control centers, and these networks open possibilities for potential attacks, unless they embed end-to-end security mechanisms and policies. Many power grid utilities deploy wireless technologies [1] and shared interconnects for cost reasons, but these solutions may weaken the security and isolation of communication in case of attacks. Even if we secure communication paths, insider attacks are possible due to spear phishing, USB, network-based malware proliferation [2], etc. As a result, it is a must for the utilities to go beyond guarding the external edges of Smart Grid SCADA networks and begin inspecting and protecting internal Smart Grid SCADA networks at all levels.

To provide end-to-end security in the network system, both the end host devices and the network need to be secured. In general purpose network such as the Internet, the end hosts usually have their own security analysis and protection mechanism. Therefore, the analyzers designed for general purpose network [3]–[6] only need to provide network analysis capability, i.e., flow/packet or application level traffic analysis. In Smart Grid SCADA network, shown in Figure 1, the end hosts are control centers and Field Devices (FDs) such as Programmable Logic Controllers (PLCs), Remote Terminal Units (RTUs) and Intelligent Electronic Devices (IEDs). Since the FDs are resource-constrained devices which do not have enough memory and computing resources to support the addition of security capabilities [2], the control centers are usually responsible for both the device status analysis and network traffic analysis. In the current architecture, data, collected by the measurement devices, is transmitted to the control center for processing and analysis. As the number of deployed measurement devices grows, it is increasingly harder for the current architecture to satisfy the real-time needs of protection and control applications. Therefore, the current Smart Grid SCADA networks cannot provide up-to-date situation awareness to the power grid operators. However, supervising device functionality, detecting networking anomalies and preventing potential problems in substations, all rely on situation awareness. The lack of up-to-date situation awareness may result in huge losses during security breaches.

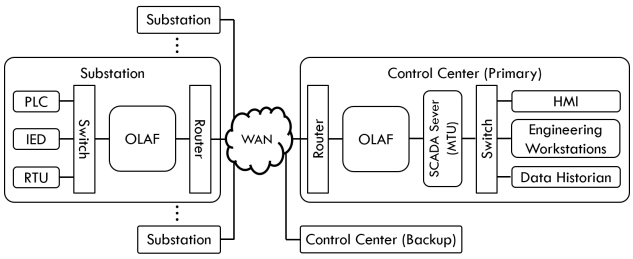


Fig. 1. Smart Grid SCADA network architecture and OLAF

Our approach to the problem in Smart Grid SCADA networks is the design of an extensible and efficient operation-level traffic analyzer, called OLAF, that has the capabilities of both flow-level network traffic analysis and operation-level device status analysis. OLAF is able to collect, aggregate and analyze the statistics in network packets from both the flow level for network traffic analysis and the operation level for device status analysis. For network traffic analysis in the flow level, OLAF is able to track which two hosts are communicating. For device status analysis in the operation

level, the analyzer is able to track status information of the utilized industrial control systems protocols (e.g., Modbus and DNP3), ongoing operations (e.g., read or write), and the targets of those operations(e.g., indexes of coils or registers that carry values of the operation). OLAF collects, aggregates and stores these meta data statistics in efficient data structure. Then, it inspects those aggregated data to perform anomaly detection.

We deploy OLAF close to the end hosts inside the Smart Grid internal network to provide up-to-date situation awareness of network traffic and operational device status to the power operator. OLAF provides the situation awareness by promptly extracting, aggregating, displaying and analyzing the control information in network packet headers as well as the encapsulated data. By providing both network and device analysis ability close to FDs, we are able to give the power operator more up-to-date view of the whole system and warn them of potential breaches more promptly.

The paper is structured as follows: We review the related work in Section II. In Section III, we introduce the network architecture of the Smart Grid control systems and describe the design challenges and our approaches. In Section IV, we present an overview of the analyzer design. Both time overhead and performance evaluation of OLAF is shown in Section V and we conlude the paper in Section VI.

II. Related Work

Traffic analyzer is the main approach to provide network traffic analysis for general purpose networks. There are many works [3]–[6] aimed at designing network profiler and traffic analyzer. However, to our knowledge, none of the existing analyzers for general purpose networks provide operation-level device analysis. And operation-level device analysis is crucial to Smart Grid SCADA networks, since compromised devices can send malicious operations or fake measurement data which could cause huge damage to the entire system.

There are also works that build analyzers for SCADA systems. Different approaches include traffic filtering systems [7], [8], Bloom-filter-based/model-base/machine-learning-based intrusion detection [9]–[12], SCADA Intelligence Gateway [13] and a fine grained analysis of packet content [14]. Our work differs from theirs in that instead of just extracting and analyzing all the features separately, we factor the features into multiple levels and store our statistics in a tree structure. We have also designed a threshold-based anomaly detection algorithm utilizing the tree structure. The tree structure not only allows us to efficiently store and access the statistics, but also gives us the ability to easily change the granularity of our analysis and inspection.

III. Network Architecture And Design Challenge

In this section, we first introduce the Smart Grid network architecture. Then we describe the design problems of the analyzer together with our approaches.

A. Network Architecture

The simplified network architecture of Smart Grid SCADA system [2] is shown in Figure 1. The communications are between the SCADA Servers or Master Terminal Units (MTUs) in control centers and FDs within the substations. The FDs can be either measurement devices such as sensors which sample grid state or control devices such as microprocessor-based controllers which can execute control commands. They are the sources of the data streams to control centers and destinations of the control streams from control centers. Each substation could have multiple FDs. Connected to multiple substations, each of the control center is responsible for processing the forwarded data and issuing control commands.

To perform analysis of the Smart Grid SCADA network traffic and device status, our analyzers are placed at the boundary of the WAN at both control center and substation ends, as it is shown in Figure 1. To be more specific, the uplink of the substation switch is connected to the input interface of the analyzer and the output interface is connected to the input of the router, making the analyzer inline to all communications. A similar type of connection is done also on the other side at the control center. Once the analyzers are physically connected inline, all traffic that was previously being communicated now transits through the analyzer and is subject to inspection and analysis.

B. Design Challenge

While designing the analyzer, there are many challenges we need to deal with. Two of them are crucial to our analyzer and are discussed as follows:

- ***Extensibility***[1]: As the Smart Grid SCADA networks evolve, smarter devices and new industrial control protocols are deployed. Designing the analyzer to be easily extensible to handle new operations of new devices using new protocols is of great importance. The software design concept, modularity, is our approach to this problem. OLAF is constructed by different modules with different functionalities (e.g., collection, aggregation, analysis). To support the analysis of new operations from new protocols, we can simply update corresponding modules or add new modules instead of redesigning the entire analyzer.
- ***Efficiency***: Since OLAF is going to deal with real-time traffic in the Smart Grid SCADA network and provide prompt analysis and inspection of the packet, time efficiency is crucial to our analyzer. The collection of statistics needs to be done at the communication line speed and the analysis also needs to be done fast enough to be meaningful to power grid operators. To be efficient, the analyzer should collect sufficient amount of statistics at the minimum cost. We achieve this by factoring the statistics into multiple levels and adjusting the number of used levels according to requirements of metadata accuracy as well as time and storage overhead.

[1]Here we only discuss the extensibility to new functionalities.

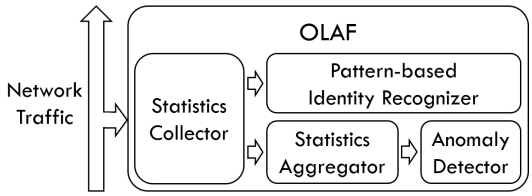


Fig. 2. Modular Structure of OLAF

IV. Analyzer Design

In this section, we present an overview of the analyzer system design. The modular structure of OLAF is shown in Figure 2. OLAF consists of 4 modules: (1)*Statistics Collector*, (2)*Statistics Aggregator*, (3)*Anomaly Detector*, (4)*Pattern-based Identity Recognizer*.

The Statistics Collector examines the network packets and collects the flow-level and operation-level statistics needed by the network traffic and device status analysis. It then provides the inputs for the Statistics Aggregator and the Pattern-based Identity Recognizer. The Statistics Aggregator aggregates the statistics and sends them to the Anomaly Detector for further analysis and anomaly detection. The Pattern-based Identity Recognizer identifies the type of the traffic source and destination by monitoring certain request and response patterns in the statistics provided by the Statistics Collector. The four modules will be described in more detail in the following subsections.

A. Statistics Collector

For each network packet that goes into the collector, there are 6 levels of statistics we want to collect, which are shown in Table I. Levels 1-2 are flow-level statistics that are used for network traffic analysis, while levels 3-6 are operation-level statistics that are used for device status analysis. Note that not all 6 levels of statistics necessarily exist for each packet. For example, a transport control packet (e.g., TCP ACK) does not have an industrial control protocol (e.g., Modbus) header and only has the upper two levels (e.g., levels 1 and 2) of the statistics. On the other hand, a Modbus request packet to read a specific register has all 6 levels of statistics. Another thing worth noticing is that the Unit ID (UID) is a protocol specific additional address used to differentiate aggregated data or devices that do not have IP addresses. For example, the substation could aggregate the measurement data from IEDs and the control center will communicate with the substation instead of each IED to collect those data. In this case, all the packets will have IP addresses of the control center and the substation. To differentiate data from different IEDs, the power operator will assign different additional addresses to each IED and use those as identifiers.

In the Statistic Collector, each packet header will go through three metadata extractors in order. The three extractors will extract the statistics of levels $\{1, 2\}$, $\{3, 4, 5\}$ and 6, respectively. The first extractor is a general one which extracts *sender* and *receiver* information. The other two, on the other hand, are protocol and device specific and are responsible for extracting $\{$*protocol*, *UID*, *function*$\}$ and *target*, respectively. Currently, we have extractors for two industrial control protocols, DNP3 and Modbus. An *item_gen* event will be triggered after the packet is processed by the last extractor, which contains all the statistics extracted by the current and all former extractors. Consider an example of a control center with IP 1.2.3.4 sending periodic Modbus *read coils* requests to a substation with IP 4.3.2.1. The control center sends the request once per minute and tries to read coil 1 from device with UID of 2. After the request packet header goes through the OLAF collector, the extractors extract the *sender* of 1.2.3.4, the *receiver* of 4.3.2.1, the *protocol* of Modbus, the *UID* of 2, the *function* of READ_COILS, and *target* of 1. Note that if new operations in new protocols need to be supported, only the last two extractors need to be updated. All the other parts can remain exactly the same, which provides significant extensibility to the analyzer.

TABLE I
5 LEVELS OF STATISTICS

Level	*Subject*
1	Sender of the packet
2	Receiver of the packet
3	Protocol that the packet uses for industrial control (e.g., Modbus or DNP3)
4	Unit ID (UID) that is used by the protocol to identify different devices
5	Function (e.g., read or write) that the packet conducts in its protocol
6	Target of the function (e.g., which coil or register)

A useful feature of our analyzer is that users can easily scale the number of levels of statistics to collect. If the user is only interested in the flow-level information, the analyzer can be configured into a general network analyzer by only collecting the upper two levels of statistics. This can largely speed up the collector and make the analyzer more efficient, since the packet needs to go through the first extractor only instead of all three in this case. If operation-level statistics is required, the analyzer collects all 6 levels of statistics and provides the capabilities of device status analysis and network traffic analysis.

B. Statistics Aggregator

The Statistics Aggregator aggregates the information of each packet and constructs a tree structure $Tree_{new}$ to store the aggregated statistics. Each tree structure $Tree_{new}$ corresponds to statistics aggregated over certain period of time T_p and each node corresponds to statistics of a specific kind of packets. An example of the data structure is shown in Figure 3. Each node (leaf and internal) in the tree includes the following fields: (1)Info string IS, (2)Accumulated info string AIS, (3)Packet count PC, (4)Average byte AB, (5)Response ratio RR (*function* level node only), (6)Response delay RD (*function* level node only). IS is the value of the corresponding statistics level. For example, IS of *sender* level is its IP address and IS of *function* level is the function name. AIS of a node, on the other hand, is constructed by ISes of nodes on the path from the root to itself. And the node stores statistical data of the kind of packets represented by its AIS. For example, the node labelled "G_1" in Figure 3 has an AIS

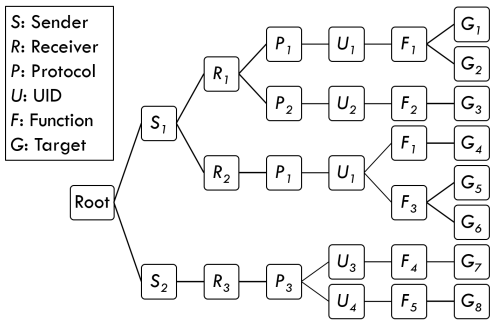


Fig. 3. Statistics structure

of "$S_1 - R_1 - P_1 - F_1 - G_1$" and therefore stands for packets that sender S_1 sends to receiver R_1 using protocol P_1 with function F_1 performed on target G_1. The other four fields are data fields used to store statistical data of that kind of packets corresponding to the node during that period T_p. PC is the total number of the packets, while AB is the average size in bytes of the packets. RR and RD only exist for *function* level nodes and are the ratio of responded request functions and the delay of the response, respectively.

In the workflow of the Statistic Aggregator, the *item_gen* event is fed into an item counter, which gets the information about the packet in the event and updates the corresponding nodes' date fields. There is also an aggregator which runs every period T_p, aggregating the results during that period as well as constructing the statistics structure. After the aggregator finishes the aggregation and construction, it triggers an *aggre_finish* event which includes the tree structure $Tree_{new}$ of that period. This event could be used for further analysis of the statistics of the network traffic such as anomaly detection which will be introduced in the following subsection.

C. Anomaly Detector

The Anomaly Detector is responsible for triggering alarms when anomalous traffic is seen in the network. There are mainly two approaches for intrusion detection: specification-base and anomaly-based. Although the anomaly-based approach has the ability to detect novel attacks, the difficulty of modelling the normal behaviour and the high false positive rate prevent it from widespread use. However, it is shown that the network traffic in SCADA systems shows much more regularity than traffic in general purpose network [14]. Therefore, the anomaly-based approach is ideal for intrusion detection in Smart Grid SCADA networks. Our current anomaly detector uses the anomaly-based approach. Specifically, it utilizes a threshold-based algorithm, named *Normal Tree*. The core idea of the algorithm is constructing a 'normal' tree $Tree_k$ which represents the normal traffic and using it as a baseline. The next tree, constructed by the Statistic Aggregator, $Tree_{new}$ is then compared to the baseline to detect any potential anomaly.

The algorithm has two phases: initialization phase and anomaly detection phase. In the trusted initialization phase, which is the first k periods with total length of kT_p, the algorithm just merges the k trees and constructs the normal tree $Tree_k$. The structure of the normal tree $Tree_k$ is similar to $Tree_{new}$ except that we store a mean value μ and a standard deviation value σ for each statistic field (PC, AB, RR, RD) in each node. So each node has IS, AIS, (μ_{PC}, σ_{PC}), (μ_{AB}, σ_{AB}) (and additionally (μ_{RR}, σ_{RR}), (μ_{RD}, σ_{RD}) for *function* level nodes).

In the anomaly detection phase, we compare $Tree_{new}$ with $Tree_k$. One node N in $Tree_{new}$ is considered to be the "same" with another in $Tree_k$ if they have the same AIS. To compare the two same nodes in two trees, we assign anomaly scores to each data field of them. Suppose the field in $Tree_{new}$ has a value of X and the corresponding field in $Tree_k$ has value μ and σ. Utilizing the Chebyshev's inequality, we define anomaly score AS as follows:

$$AS(X, \mu, \sigma) = \begin{cases} 1 - \frac{\sigma^2}{|X-\mu|^2} & \text{if } |X - \mu| > \sigma \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

The anomaly score is in the range $[0, 1]$ and a higher score represents more abnormal behavior. The algorithm then compares the score with a predefined threshold θ and decides whether to trigger an alarm or not. The anomaly detection phase of the algorithm is shown in Algorithm 1.

Algorithm 1 Normal Tree Algorithm

procedure ANOMALYDETECT($Tree_{new}, Tree_k, \theta$)
  Traverse $Tree_{new}$ in pre-order.
  for each node N in $Tree_{new}$: **do**
    if N also exists in $Tree_k$ **then**
      Use Equation 1 to calculate AS for each data field of N and compare them with θ. Continue to traverse N's children.
    else if N does not exist in $Tree_k$ **then**
      Assign $AS = 1$ to N instead of each data field and compare it with θ. Stop traversing N's children.
    end if
  end for
  Traverse $Tree_k$ in pre-order.
  for each node N in $Tree_k$: **do**
    if N does not exist in $Tree_{new}$ **then**
      Create a dummy node N with all data fields set to zero. Then use Equation 1 to calculate AS for each data field and compare them with θ. Stop traversing N's children.
    end if
  end for
end procedure

Consider our previous example of periodic *read coils* requests. Suppose this is the only traffic in the network and the substation never responds. We choose $T_p = 10min$, $k = 10$ and $\theta = 0.5$. The normal tree $Tree_k$ is represented by $Root_1 - S_1 - R_1 - P_1 - U_1 - F_1 - G_1$. All nodes have data fields $(\mu_{PC}, \theta_{PC}) = (10, 0)$, $(\mu_{AB}, \theta_{AB}) = (64, 0)$ and F_1 has additional data field $(\mu_{RR}, \theta_{RR}) = (0, 0)$. Now suppose the control center starts to send write coil requests instead of read coils requests to the same device and target

at the same frequency. The next $Tree_{new}$ is represented by $Root_2 - S_2 - R_2 - P_2 - U_2 - F_2 - G_2$. The algorithm traverses $Tree_k$ and $Tree_{new}$ simultaneously in pre-order. It first finds that $Root_1$ and $Root_2$ have the equal AIS and therefore are same nodes. The calculated AS for both PC and AB are zeros, which are less than $\theta = 0.5$. Hence the algorithm does not trigger any alarm and continues to compare their children. Similarly, the algorithm compares S_1 with S_2, R_1 with R_2, P_1 with P_2, U_1 with U_2 in order and triggers no alarm. Then the algorithm finds that no node in $Tree_k$ has the same AIS with that of F_2, so it assigns $AS = 1$ to F_2. Since $AS = 1 > 0.5 = \theta$, it triggers an alarm. The algorithm also finds that no node in $Tree_{new}$ has the same AIS with that of F_1, so it creates a dummy node F_0 with all data fields set to zero and compares F_0 with F_1. $AS_{PC} = AS_{AB} = 1 > 0.5 = \theta$, hence the algorithm triggers two alarms.

The most criticized issue with the anomaly-based approach is the high false positive rate. There are three parameters in our analyzer that we can increase to reduce false positives: period time T_p, number of training periods k, and the anomaly score threshold θ. But they need to be carefully tuned since there are trade-offs and restrictions. Because we only detect anomalies at the end of each period, the period time T_p decides the maximum anomaly detection delay! We need to keep it small to provide prompt detection! A longer training phase has a larger training cost, and makes it harder to keep the training set clean. Increasing the threshold will make the detection algorithm less sensitive, which might decrease the detection rate. OLAF also has a feedback loop between the anomaly detector and the power grid operator. If the operator finds one alarm to be false positive, he or she can give feedback to the detector so that the detector can adapt to avoid the same mistake. Our current naive feedback mechanism just increases the standard deviation in the corresponding data field in the normal tree. More sophisticated methods will be explored in the future work.

D. Pattern-based Identity Recognizer

The objective of the Pattern-based Identity Recognizer is to identify the device type of the traffic source and/or destination by monitoring certain request and response patterns in the traffic statistics.

The recognizer consists of a request-response coupler and a recognition rule matcher. The request-response coupler analyzes the packet statistics in the *item_gen* event. Based on the statistics, it couples each pair of request and response and constructs a variable which consists of the protocol and both the request and response function codes. For each pair of request and response, this variable is checked by the recognition rule matcher to see whether it matches the function pairs given by the recognition rules or not. If a match is found, the identities of the requester and/or the responder are also given by the corresponding matched rule and output by the matcher.

V. Evaluation

In this section, we present the evaluation of OLAF in terms of time overhead and detection ability. We use a network analysis framework, called Bro [15], to implement our analyzer. All experiments run on a simulated Modbus trace set. The trace set includes a baseline traffic flow and some injected anomalies. In the baseline traffic, one Modbus master (control center) sends periodic operations to 10 Modbus slaves (FDs). The valid operations in the baseline traffic and the frequencies of them are shown in Table II.

TABLE II
Frequencies of valid operations in the baseline traffic

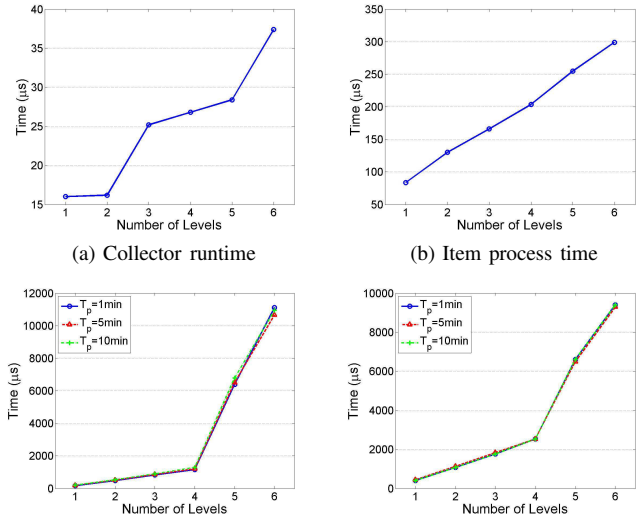
Operation	Function code	Frequency
Read Coils	01	Every 5 seconds
Read Discrete Inputs	02	Every 5 seconds
Read Holding Registers	03	Every 5 seconds
Read Input Registers	04	Every 5 seconds
Write Single Coil	05	Every 30 seconds
Write Single Register	06	Every 30 seconds
Write Multiple Coils	15	Every 5 minutes
Write Multiple Registers	16	Every hour

A. Time Overhead Evaluation

Since our analyzer is inline to all traffic going through, we need to evaluate its processing time overhead to make sure it can handle real time traffic. We run OLAF on a 64-bit Ubuntu machine with 8 Intel i7-2600 3.40GHZ CPUs and 3.7GiB memory. In each of the following experiments, we run the corresponding modules on the same trace set for 5 rounds and take the average.

For the Statistics Collector, we measure the processing time of each individual packet and define it to be the total runtime of the collector. For the Traffic Statistics Counter, we are interested in the item process time and the aggregation time. Item process time is the time for the item counter to process the *item_gen* event and update the corresponding date fields and is measured per *item_seen* event. Aggregation time, which is measured per aggregation period, is the time for the aggregator to aggregate the data fields and construct the current tree structure. For the Anomaly Detector, the time to run the Normal Tree algorithm for one period is denoted by the anomaly detection time. Note that the total real-time processing time of each packet is the sum of collector runtime and item process time. And since the aggregation and anomaly detection is only done once for each aggregation period, they are not subject to the real-time overhead requirement.

Figure 4 shows the above four kinds of time overhead with different number of levels and aggregation period time. We can see that even if we collect statistics from all 6 levels, the total real-time processing time of each packet is still below 350 μs. And this is short enough for the packets to be processed in communication line speed. Moreover, T_p does not affect the aggregation time and anomaly detection time much since the traffic follows certain patterns. Most importantly, reducing the number of levels has a significant effect on the decrease of the time overhead of different modules. Therefore, always using the least necessary number of levels can save non-negligible



(a) Collector runtime (b) Item process time

(c) Aggregation time (d) Anomaly detection time

Fig. 4. Time overhead with different levels and aggregation period

amount of time and produces the highest processing speed. In this way, efficiency can be achieved by the analyzer.

B. Detection Ability Evaluation

To evaluate OLAF's anomaly detection ability, we inject anomalies in different levels to the baseline traffic. The injected anomalies are listed in Table III. We fix the period time T_p to be 10 minutes and the anomaly detection phase to be 6 hours, and change the training phase length k and anomaly score threshold θ. We also evaluate the analyzer in both feedback off and on situations. OLAF is able to detect all nine anomalies in all cases, which means it never miss any anomaly. And we define ratio of number of false alarms to number of total node checked to be false alarm rate. The false alarm rate in different cases are listed in Table IV. We can see that increasing training time and anomaly score threshold both reduce false positives. And the feedback loop also has a huge effect on decreasing the false positive rate.

TABLE III
INJECTED ANOMALIES

Level	Injected anomaly
Flow level	Add one new Modbus master to send a set of operations to one Modbus slave
	Drop one Modbus slave
	Send a bunch of ICMP packets to one Modbus slave
	Increase the response delay of one Modbus slave from 30ms to 200ms
	Increase the packet drop rate of one Modbus slave from 0 to 30%
Operation level	Stop sending some operations to one Modbus slave
	Send several new operations to one Modbus slave
	Change the sending frequency of some operations for one Modbus slave
	Change the targets of one operation for one Modbus slave

VI. Conclusion

In this paper, we present an extensible and efficient operation-level traffic analyzer framework, called OLAF, for Smart Grid SCADA networks to provide network traffic analysis and device status analyses. By collecting, aggregating and analyzing statistics in both flow level and operation level on the communication within the internal network, OLAF increases the situation awareness and security of the control system. Our results are strongly encouraging to place the extensible and efficient analyzer in Smart Grid SCADA networks.

TABLE IV
FALSE ALARM RATE WITH DIFFERENT PARAMETERS

Training Time		4 hours		8 hours	
Feedback		Off	On	Off	On
Threshold θ	0.9	4.16%	1.07%	3.01%	0.86%
	0.99	2.35%	0.62%	0.10%	0.06%
	0.999	2.26%	0.56%	0.05%	0.03%

Acknowledgment

This material is based upon work supported by the Department of Energy under Award Number DE-OE0000676.

References

[1] H. Jin, S. Uludag, K.-S. Lui, and K. Nahrstedt, "Secure data collection in constrained tree-based smart grid environments," in *Smart Grid Communications (SmartGridComm), 2014 IEEE International Conference on*. IEEE, 2014, pp. 308–313.

[2] K. Stouffer, J. Falco, and K. Scarfone, "Guide to industrial control systems (ics) security," *NIST special publication*, vol. 800, no. 82, pp. 16–16, 2011.

[3] T. McGregor, H.-W. Braun, and J. Brown, "The nlanr network analysis infrastructure," *Communications Magazine, IEEE*, vol. 38, no. 5, pp. 122–128, May 2000.

[4] W. Erhard, M. Gutzmann, and H. Libati, "Network traffic analysis and security monitoring with unimon," in *High Performance Switching and Routing, 2000. ATM 2000. Proceedings of the IEEE Conference on*, 2000, pp. 439–446.

[5] D. Keim, F. Mansmann, J. Schneidewind, and T. Schreck, "Monitoring network traffic with radial traffic analyzer," in *Visual Analytics Science And Technology, 2006 IEEE Symposium On*, Oct 2006, pp. 123–128.

[6] M. Rahman, Z. Khalib, and R. Ahmad, "A portable network traffic analyzer," in *Electronic Design, 2008. ICED 2008. International Conference on*, Dec 2008, pp. 1–6.

[7] I. N. Fovino, A. Coletta, A. Carcano, and M. Masera, "Critical state-based filtering system for securing scada network protocols," *IEEE Transactions on industrial electronics*, vol. 59, no. 10, pp. 3943–3950, 2012.

[8] B.-K. Kim, D.-H. Kang, J.-C. Na, and T.-M. Chung, "Abnormal traffic filtering mechanism for protecting ics networks," in *2016 18th International Conference on Advanced Communication Technology (ICACT)*. IEEE, 2016, pp. 436–440.

[9] S. Parthasarathy and D. Kundur, "Bloom filter based intrusion detection for smart grid scada," in *Electrical & Computer Engineering (CCECE), 2012 25th IEEE Canadian Conference on*. IEEE, 2012, pp. 1–6.

[10] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner, and A. Valdes, "Using model-based intrusion detection for scada networks," in *Proceedings of the SCADA security scientific symposium*, vol. 46. Citeseer, 2007, pp. 1–12.

[11] J. M. Beaver, R. C. Borges-Hink, and M. A. Buckner, "An evaluation of machine learning methods to detect malicious scada communications," in *Machine Learning and Applications (ICMLA), 2013 12th International Conference on*, vol. 2. IEEE, 2013, pp. 54–59.

[12] L. A. Maglaras and J. Jiang, "Intrusion detection in scada systems using machine learning techniques," in *Science and Information Conference (SAI), 2014*. IEEE, 2014, pp. 626–631.

[13] B. Panja, J. Oros, J. Britton, P. Meharia, and S. Pati, "Intelligent gateway for scada system security: A multi-layer attack prevention approach," in *2015 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)*. IEEE, 2015, pp. 1–6.

[14] D. Hadziosmanovic, D. Bolzoni, S. Etalle, and P. Hartel, "Challenges and opportunities in securing industrial control systems," in *Complexity in Engineering (COMPENG), 2012*. IEEE, 2012, pp. 1–6.

[15] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer networks*, vol. 31, no. 23, pp. 2435–2463, 1999.