Efficient Framework to Mitigate DDoS Attacks and Response System

[1]Saravanan S, [2]P.Siva kumar

[1]Assistant Professor, Department of IT
PSNA College of Engineering & Technology,
Dindigul, Tamil Nadu, India
ssaravananme@gmail.com

[2]Assistant Professor, Department of IT
PSNA College of Engineering & Technology,
Dindigul, Tamil Nadu, India
sivakumar.paulraj@gmail.com

Abstract: ***A Distributed Denial-of-Service (DDoS) attack is one in which a multitude of compromised systems attack a single target, thereby causing denial of service for users of the targeted system. The flood of incoming messages to the target system essentially forces it to shut down, thereby denying service to the system to legitimate users .The nature of the threats posed by DDoS attacks on large networks, such as the Internet, demands effective detection and response methods. These methods must be deployed not only at the edge but also at the core of the network. This paper presents methods to identify DDoS attacks by computing entropy and frequency-sorted distributions of selected packet attributes. The DDoS attacks show anomalies in the characteristics of the selected packet attributes. The detection accuracy and performance are analyzed using live traffic traces from a variety of network environments ranging from points in the core of the Internet to those inside an edge network. The results indicate that these methods can be effective against current attacks and suggest directions for improving detection of more stealthy attacks. We also describe our detection-response prototype and how the detectors can be extended to make effective response decisions.***

Keywords: Distributed Denial of Service; Entropy; Edge network; Stealthy Attacks;

1. Introduction

Powerful DDoS toolkits are available to potential attackers, and essential networks are ill prepared for defence. The security community has long known that DDoS attacks are possible, but only in the past three years have such attacks become popular with hackers. As ominous as the threat is today, it will only worsen as tools are built to evade defences. Soon, DDoS floods will appear that are difficult to distinguish from legitimate traffic, and packet rates from individual flood sources will be low enough to escape notice by local administrators. To meet the increasing need for detection and response, researchers face these major issues:

- A stand-alone router on the attack path should automatically recognize that the network is under attack and adjust its traffic flow to ease the attack impact downstream.
- The detection and response techniques should be adaptable to a wide range of network environments, preferably without significant manual tuning.
- Attack detection should be as accurate as possible.
- False positives can lead to inappropriate responses that cause denial of service to legitimate users. False negatives result in attacks going unnoticed.
- Attack response should employ intelligent packet discard mechanisms to reduce the downstream impact of the flood while preserving and routing the non-attack packets [4].
- The detection method should be effective against a variety of attack tools available today and also robust against future attempts by attackers to find detection.

2. Related work

These are demanding goals, but we contend that there are several reasons to believe that satisfactory detection and response methods can be designed. DDoS traffic generated by today's tools often has packet crafting characteristics that make it possible to distinguish from normal traffic [3]. For example, in some configurations the Stacheldraht attack tool crafts packets so that the source port is random and the destination port is sequentially increased from one packet to the next [1][2][10]. Future DDoS tools may include improvements to packet crafting. However, we claim that these tools are unlikely to model legitimate traffic closely enough to produce crafted packets that do not distort statistical measurements of the composition of the traffic. Our hypothesis is that relatively simple statistical measures can be used to discriminate DDoS traffic from legitimate traffic in core routers with sufficient accuracy to mitigate the effect of the attack downstream.

3. Proposed Work

One common method of attack involves saturating the target machine with external communications requests, such that it cannot respond to legitimate traffic, or responds so slowly as to be rendered effectively unavailable. Such attacks usually lead to a server overload. In general terms, DDoS attacks are implemented by either forcing the targeted

computer(s) to reset, or consuming its resources so that it can no longer provide its intended service or obstructing the communication media between the intended users and the victim so that they can no longer communicate adequately. We have imposed some significant constraints on our DDoS defense development: no explicit coordination (e.g., pushback [5][6][7]) between defending network components, no built-in knowledge of applications or protocols, and no instrumentation at end hosts. These approaches are being actively explored in other research, and we believe that the techniques described here can complement these others in a comprehensive DDoS defense solution.

3.1 Detection Algorithms

Our detection algorithms measure statistical properties of specific fields in the packet headers at various points in the Internet. For instance, if a detector captures 1000 consecutive packets at a peering point and computes the frequency of occurrence of each unique source IP address in those 1000 packets, then the detector will have a model of the distribution of the source address. Further computations with this distribution allow us to measure the randomness or uniformity of the addresses as well as the "goodness-of-fit" of the distribution with respect to prior measurements.

3.2 Entropy

Let an information source have *n* independent symbols each with probability of choice *pi*.

$$H = -\sum_{i=1}^{n} \square.\ P_i \log_2 P_I \qquad (1)$$

Hence, entropy can be computed on a sample of consecutive packets. Comparing the value for entropy of some sample of packet header fields to that of another sample of packet header fields from the same peering point provides a mechanism for detecting changes in the randomness. We have observed through experimentation that while a network is not under attack, the entropy values for various header fields each fall in a narrow range. While the network is under attack with current attack tools, these entropy values exceed these ranges in a detectable manner. The algorithm to compute entropy can be optimized to perform only a few simple computations per packet. In our implementation, the entropy of a source will be calculated through a sliding window of fixed width, *W*. The probability value *pi* in this algorithm is actually the frequency of occurrence of each unique symbol divided by the total number of symbols in the sample. The process of computing entropy of *W* packets is as follows:

1. Compute the entropy of the first W packets with reference to a specific header parameter (e.g. source IP address).
2. Isolate the term in the summation corresponding to the probability of the first symbol in the window (label this symbol with *i*=1) and also the value for the corresponding probability (pi-1).
3. Slide the window so the new first term was previously the second term and the next *W*-1 consecutive terms are contained in the window.
4. Isolate the term in the summation corresponding to the probability of the symbol acquired from shifting the window.
5. Subtract off the terms isolated in steps 2 and 4 from the value computed in step 1.
6. Re-compute the affected probabilities for the current window of data. That is, re-compute *pi-1* and the probability of the symbol that was added by sliding the window.
7. Using the values computed in step 6, add the two terms missing from the entropy summation back in and compare this new entropy value to the previous entropy computations.
8. Repeat steps 2-7 to determine subsequent entropy values.

A sophisticated attacker would likely attempt to defeat the detection algorithm by creating stealthy traffic floods that mimic the legitimate traffic the detector would expect. An attacker who knew that the entropy of various packet attributes was being monitored could build an attack tool that generates floods with tunable entropy levels. Through guesswork, penetration, or trial and error, the attacker could determine typical entropy levels seen at the detector and tune the flood to match. This may not be as easy as it sounds, particularly if there are multiple detectors deployed between the flood sources and the targets, as the typical entropy values seen by detectors in different network environments are likely to differ. The window size, W, is a tunable parameter that controls how much smoothing of short-term fluctuations the detector will do. Increasing W will reduce the variation in entropy and may reduce the rate of false positives resulting from brief and presumably insignificant anomalies. However, W should be kept small enough that attacks are detected quickly. We have found that a window size of 10,000 packets is a reasonable compromise in the network environments we have explored.

3. 3 Chi Square Statistic

Pearson's chi-square (χ^2) Test is used for distribution comparison in cases where the measurements involved are discrete values. For example, it could be used to test the distribution of TCP SYN flag values (0 or 1) or protocol numbers. The test works best when the number of possible values is small. In particular, a rule of thumb is that the expected number of packets in a sample having each possible value be at least five. However, this can often be achieved through "binning", that is combining a set or range of possible values and treating them as one. For example, the chi-square test can be applied to service ports by considering four values: HTTP, FTP, DNS, and other. Similarly, packet lengths can be binned into ranges such as 0-64 bytes, 65-128 bytes, 129-255 bytes, etc. For a sample of *N* packets, let *B* be the number of available bins. Define *Ni* as the number of packets whose value falls in the *i*th bin and *ni* as the *expected* number of packets in the *i*th bin under the typical distribution. Then the chi-square statistic is computed as follows:

$$\chi^2 = \sum_{i=1}^{B} \frac{Nt - Yt}{Nt} \qquad (2)$$

When the *Ni* and *ni* values are large and the *N* measurements are independent and drawn from the expected distribution, this value follows the well-known

chisquare distribution with *B-1* degrees of freedom. These assumptions (in particular, independence) do not typically hold for packet field values even under normal conditions. Hence, comparison with the chi-square distribution is of limited utility. However, the chisquare statistic does provide a useful measure of the deviation of a current traffic profile from the baseline.

A current-traffic profile, mapping packet attribute values to frequencies, is maintained as follows:

1. For each packet that arrives, extract the value, v, of the desired attribute (e.g., source address).
2. Apply exponential decay to the stored frequency for v based on its age (time since last update).
3. Increment the frequency for v and store the current time (or packet count) as its last-update time.

Periodically, this current-traffic profile is compared with a baseline profile using the chi-square statistic, as follows:

1. Apply exponential decay to the stored current traffic frequencies, as above.
2. Group the attribute values into bins based on frequency. For example, the 16 most common values might go in one bin, the next 64 in another, the next 256 in another, and the rest in another.
3. Calculate the total frequency for each bin.
4. Calculate the chi-square statistic, comparing these bin-frequency totals with the bin frequency values in the baseline profile.

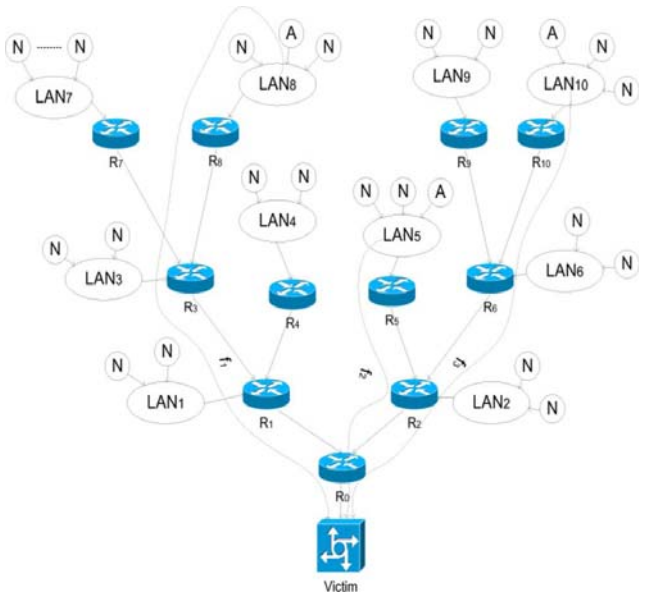

Figure 1: Architecture diagram

The baseline profile can be maintained as decaying averages of the current-traffic bin frequencies. Each time the current-traffic bin frequencies are computed, the average is updated as follows:

1. Exponential decay is applied to the stored bin frequency averages, using a significantly longer half-life than is used for the current-traffic profile.
2. The new set of bin frequencies is multiplied by and the result is added to the decayed average. The user can tune the detector by modifying the following parameters: traffic profile half-life, baseline profile half-life, bin definitions and hash function range. Values in the current-traffic profile whose frequencies decay below a certain threshold can be purged without substantially affecting the chi-square computation. This purging reduces memory consumption and processing requirements. For packet attributes such as IP addresses that have a very large range, a hash of the attribute's value may be used instead of the value itself in order to reduce memory consumption and processing requirements in the worst case (many distinct values). When the baseline frequency value for a given bin is very low, the chi-square statistic may be excessively influenced by that bin's value. Ideally, the bins will be defined such that this is unlikely, but as a fallback, low-value bins can be automatically merged with adjoining bins prior to computing the chi-square statistic.

It is unlikely that an outside attacker without access to the detector itself or a large fraction of its network neighbors will know the exact characteristics of network traffic typically seen by the detector. Therefore, we hypothesize that the attack traffic will differ from typical traffic in measurable ways.

4. Result and Analysis

4.1 Detector Evaluation*:*

The chi-square and entropy detectors were built as Snort pre processors, operating on every IP datagram received by Snort prior to stream reassembly and other packet manipulation. The two detectors can be individually enabled and configured in the snort. Configuration file and can trigger alarms through Snort's modular alerting facility. In addition to issuing alerts, these plug-ins record data to log files in the Snort log directory. The entropy detector logs periodically computed entropy values for each packet attribute specified in the initialization file (e.g., source and destination IP addresses and TCP/UDP ports, datagram length, and TCP window size). The chi-square detector logs the periodically computed chi-square statistics for each of the specified packet attributes, along with the current and baseline bin frequency values used to compute those statistics. This data can be useful for manual or automatic detector tuning and alert threshold setting [8][9].

4.1.1 Network Trace Data

A critical element of evaluating these detectors is exposing them to traffic from a variety of network environments. This allows us to determine how stable the traffic statistics monitored by the detectors are in those environments, and how effectively the detectors can identify DDoS attack traffic in different contexts.

For this purpose, we obtained several publicly available network traces as well as some traces collected specifically for our experiments. These traces are not known to contain substantial DDoS attacks, so we treat them as consisting of legitimate traffic. To test the effects of DDoS attacks, we simulate these attacks by overlaying the kind of attack traffic generated by some existing DDoS attack tools onto the traces at various concentrations [10].

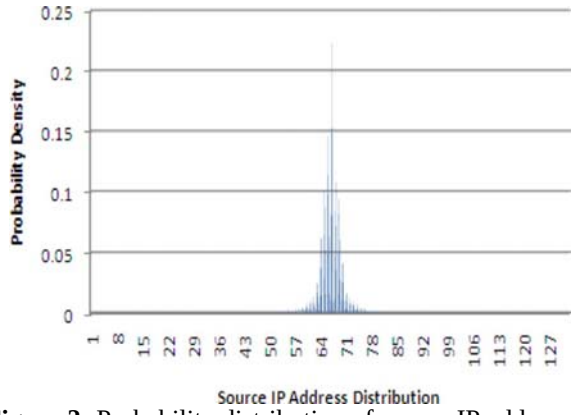

Figure 2: Probability distribution of source IP address in low-rate DDoS attack (only attack traffic) scenario

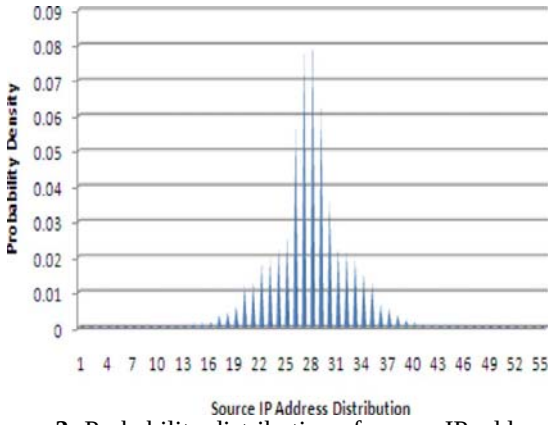

Figure 3: Probability distribution of source IP address in normal network traffic (attack free) scenario

Ideally, we would make use of traces containing identifiable periods during which actual DDoS attacks were in progress, but few of these are publicly available. The traces used were drawn from a variety of network environments, as described below, and most have IP addresses that have been transformed via an unknown but one-to-one function for privacy purposes. This address re-mapping is irrelevant to the currently implemented detectors, since they make no assumptions about relationships between different IP addresses.

4.2 Detector Performance

Since we are proposing to use these detection methods in high-speed core routers, it is imperative that they have low computational cost, especially for the operations that must be carried out for each packet. The prototype Snort detector implementation exhibits adequate performance for its purposes: on a 1GHz Pentium-III-based machine, a Snort process running a single chi-square detector observing source addresses can process 240,000-270,000 packets per second (pps) offline. (The Snort infrastructure without any plugins can handle 435,000 pps.) Adding chi-square detectors for four additional packet attributes brings performance down to around 100,000 pps. A single-attribute entropy detector can manage about 294,000 pps, while adding six others yields 130,000 pps. These speeds are roughly in the OC3 range. Improving performance is a primary goal of future detector development. We expect to achieve improved performance by implementing some optimizations that approximate the true frequency profile while reducing or eliminating floating-point operations in the packet-handling code. Most of the partitioning and computation of chi-square and entropy values can be handled asynchronously in background processes that should not impede the packet-handling fast path.

4.3 Response

Our defence approach involves response modules that use a characterization of the attack provided by the detection module to take defensive measures. The response module classifies individual packets as benign or suspect based on the attack characteristics provided by the detector. Once identified, the suspect packets are subjected to rate limiting or packet-filtering methods based on the intensity of the attack or pre-defined response policies. In the case of stealthy DDoS attacks, the response module should communicate with the detector and share the data structures and statistical models maintained by the detector to identify the attack packets with high confidence; the prototype described below does not yet offer such coordination[11][12].

4.3.1 Prototype DDoS Response Module

The current response prototype is implemented on a Linux router as a kernel module. It uses netfilter and Linux Advanced Routing and Traffic Control (LARTC) to filter and rate-limit packets [4]. An API is provided to take alerts from the detection module and generate filter rules to be issued to the response module. We have also produced an extension to the Linux iptables mechanism that provides similar functionality, for better integration with iptables-based router/firewall configurations. Currently, the response module implements three packet-filtering rules. They are constant, random and allow. These filter rules are automatically generated by the Snort-based DDoS detector when it issues an alert. When the detector module detects a DDoS flood, it uses its detection algorithms and the statistical models to characterize the DDoS packets. The characteristics of the DDoS packets are used to form one of the three packet filter rules. The detector can then insert the packet filtering rules using the /proc file system interface. These rules will filter out the DDoS attack packets. Once the detector determines that the attack has subsided, it can remove the appropriate filtering rules.

(a) Constant Filter Rule

A constant filter rule is used to drop packets that match the values specified in the rule to the values of the protocol header fields in the packet. This filter rule can be applied to the IP header fields, TCP source and destination ports, UDP source and destination ports and ICMP type and code fields. For example, the rule "const {daddr=10.1.1.10 protocol= 6 dport=80 sport=31137*};"* will drop TCP packets going to the particular destination IP address 10.1.1.10 and TCP destination port 80 from the TCP source port 31137. A number of such *constant* filter rules can be applied to the response module. If none of the *constant* filter rules match the values of the header fields, the packet is allowed to pass.

(b) Random Filter Rule

A number of DDoS attack tools create packets with random values in the header fields. That is, a random number generator is used to assign a value to certain fields in the header. In such a case, a random filter rule can be specified to drop packets with random values in the header field[13][14]. The random filter rule can be applied to the IP header fields, TCP source and destination ports, UDP source and destination ports, and ICMP type and code fields. Random filter rules can be applied in the following cases:

- A packet has only one random item in the header field. For example, the rule "rand {saddr};" will drop packets with only random source IP addresses.
- A packet with more than one random item in the header field. For example, the rule "rand {tot_len saddr protocol};" will drop packets with random values for total IP packet length, source IP address and IP protocol.

5. Conclusion and Future Extensions

The focus thus far has been on detection and response algorithms and the implementation of these algorithms in software. At issue is whether these algorithms can reliably detect and respond to DDoS attacks. Against today's relatively unsophisticated DDoS toolkits, our prototype detector is able to determine that the network is under attack and deploy accurate filtering rules.

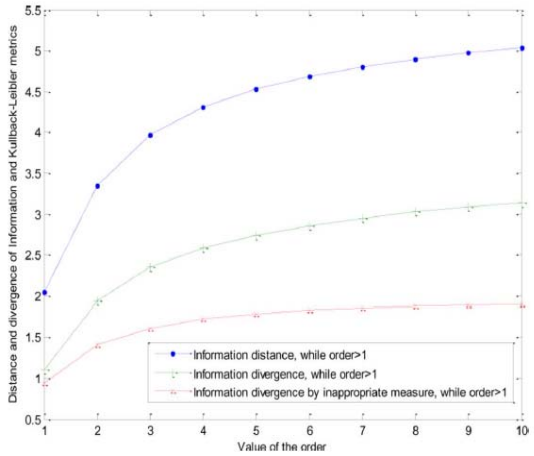

Figure 4: Variations of information distance and divergence as well as divergence by inappropriate measure along with the value of the order $(\alpha > 1)$, the Kullback–Leibler distance and divergence as well as the divergence by inappropriate Measure while $\alpha = 1$.

The filtering effort is immediate and reduces the impact of the attack downstream almost instantly. Because baseline measurements and thresholds can be established automatically, and because detectors can generate filtering rules automatically based on the traffic statistics they gather, the system is adaptable to a wide range of network environments with minimal manual tuning.

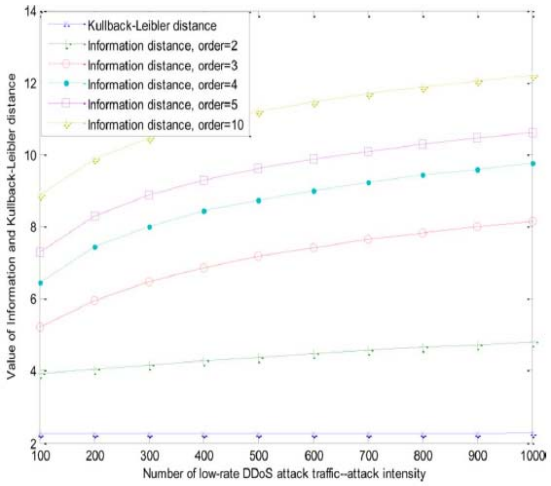

Figure 5: Variations of distance of the information and Kullback – Leiblermetrics in increasing DDoS attack intensity quickly

While our initial goal was to provide effective defence against existing DDoS tools, we are continuing to explore techniques for better defence against future stealthy attacks. Future research and development will focus on tighter integration of detection and response modules. In the current implementation, detectors generate concise recommended rules for responders to impose, and there is no further detector/responder coordination. In a more tightly coupled detection/response system, the individual packet classification decisions made by the responder could make use of the rich data structures maintained by the detector.

References

[1] D. Dittrich, "The 'Stacheldraht' Distributed Denial of Service Attack Tool", http://staff.washington.edu/dittrich/ misc/stacheldraht.analysis, 1999.

[2] C. Faloutsos, M. Faloutsos, and P. Faloutsos, "On Power-Law Relationships of the Internet Topology," Proc. Of ACM SIGCOMM, Aug. 1999, pp. 251-262.

[3] T.M. Gil, M.A. Poletto and E.W. Kohler, Jr. "Statistics Collection for Network Traffic", United States Patent Application, March 21, 2002.

[4] B. Hubert, "Linux Advanced Routing and Traffic Control HOWTO", http://lartc.org/howto/.

[5] D. Knuth, The Art of Computer Programming: Seminumerical Algorithms, Third edition, Vol. 2, Addison- Wesley, Reading, Massachusetts, 1997.

[6] M.V. Mahoney and P.K. Chan, "Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks", SIGKDD '02, Edmonton, Alberta, Canada, July 23-26, 2002, pp. 376-385.

[7] R. Manajan, et al., "Controlling High Bandwidth Aggregates in the Network", SIGCOMM Computer Communications Review, 32(3), July 2002.

[8] D. Moore, G. Voelker, and S. Savage, "Inferring Internet Denial-of-Service Activity", Proceedings of USENIX Security Symposium 2001, pp. 9-22.

[9] E. Mouw, "Linux Kernel Procfs Guide", http://www.kernelnewbies.org/documents/kdoc/procfs-guide/ lkprocfsguide.html.

[10] Netflood Infosec Tools & Resources, Source Code to Stacheldraht, http://netflood.net/files/Dos/DDoS.

[11] O. Pomerantz, "Linux Kernel Module Programming Guide", http://www.tldp.org/LDP/lkmpg/mpg.html.
[12] P.A. Porras, and P.G. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances," Proceedings of the National Information Systems Security Conference (NISSC), October 1997, pp. 353-365.
[13] M. Roesch, (March 2002), "Snort Users Manual: Snort Release 1.8.5", http://www.snort.org/documentation.html, March 2002.
[14] M. Roesch, "Snort - Lightweight Intrusion Detection for Networks" Proceedings of the 13th Systems Administration Conference (LISA'99), USENIX Association, 1999, pp. 229-238, http://www.snort.org/docs/lisapaper.txt.

Author Profile

Mr. S. Saravanan received his B.Tech Degree in Information Technology from RVS College of Engineering & Technology, Dindigul and also completed his M.E. Computer Science & Engineering in P.S.N.A college of Engineering and Technology, Dindigul, Tamilnadu. He is currently working as a Assistant Professor in Department of Information Technology, PSNACET, Dindigul. His research interests include mesh networks, ad-hoc networks, network security, cloud computing.

Mr. P. SivaKumar received his B.E Degree in Computer Science from KCG College of Technology, Chennai and also completed his M.E. Computer and Communication Engineering in P.S.N.A college of Engineering and Technology, Dindigul, Tamilnadu. He is currently working as a Assistant Professor in Department of Information Technology, PSNACET, Dindigul. His research interests include Vehicular ad-hoc networks, cloud computing..