IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 1 Issue 3, May 2014.

www.ijiset.com

ISSN 2348 - 7968

THE REDUCTION OF VAMPIRE ATTACKS ON DISTANCE VECTOR ROUTING PROTOCOLS FOR WIRELESS ADHOC SENSOR NETWORKS

A. Sabitha bharathi[1], A.sivasakthivel[2],

[1]PG Scholar, Applied Electronics, Arunai Engineering College, Thiruvannamalai, India,

[2]Assistant professor, Department of ECE, Arunai Engineering College, Thiruvannamalai, India,

Abstract- Vampire attacks are a new class of resource consumption attacks that permanently disable ad hoc wireless sensor networks by depleting nodes' battery power. A node is permanently disabled once its battery power is exhausted. These "Vampire" attacks are not specific to any specific protocol, but rather rely on the properties of many popular classes of routing protocols. We find that all examined protocols are susceptible to Vampire attacks, which are devastating, difficult to detect, and are easy to carry out using as few as one malicious insider sending only protocol-compliant messages. In the worst case, a single Vampire can increase network-wide energy usage by a factor of O(N), where N in the number of network nodes. Some methods were discussed to mitigate these types of attacks, including a new proof-of-concept protocol that provably bounds the damage caused by Vampires during the packet forwarding phase by using the Dynamic Source Routing (DSR). The effect of Vampire attacks on distance vector routing protocols using a small number of weak adversaries is showed, and their attack success on a randomly generated topology of 30 nodes is measured. Simulation results show that depending on the location of the adversary, network energy expenditure during the forwarding phase increases.

Index Terms- *Dynamic Source Routing (DSR), Denial of service, security, routing, ad hoc networks, sensor networks, wireless networks.*

I. INTRODUCTION.

WIRELESS SENSOR NETWORKS

Sensor networks are the key to gathering the information needed by smart environments, whether in buildings, utilities, industrial, home, shipboard, transportation systems automation, or elsewhere. Recent terrorist and guerilla warfare countermeasures require distributed networks of sensors that can be deployed using, e.g. aircraft, and have self-organizing capabilities. In such applications, running wires or cabling is usually impractical. A sensor network is required that is fast and easy to install and maintain. Wireless sensor networks satisfy these requirements. Desirable functions for sensor nodes include: ease of installation, self-identification, self-diagnosis, reliability, time awareness for coordination with other nodes, some software functions and DSP, and standard control protocols and network interfaces A wireless sensor network (WSN) consists of spatially distributed autonomous sensors to monitor physical or environmental conditions, such as temperature, sound, pressure, etc. and to ooperatively pass their data through the network to a main location. The more modern networks are bi-directional, also enabling control of sensor activity. The development of wireless sensor networks was motivated by military applications such as battlefield surveillance etc.

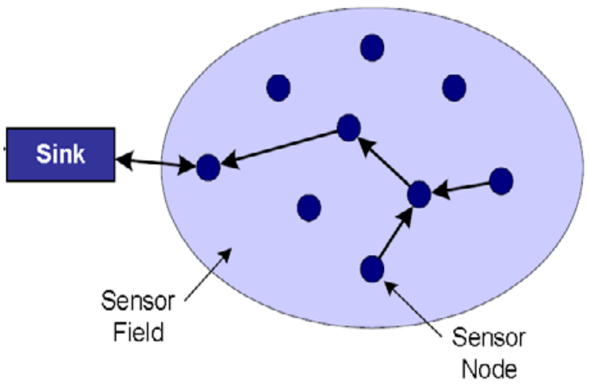


Fig 1 Typical wireless sensor network example

The main reason is that the set of assumptions underlying previous work has changed dramatically. Most past distributed systems research has assumed that the systems are wired, have unlimited power, are not real-time, have user interfaces such as screens and mice, have a fixed set of resources, treat each node in the system as very important and are location independent. In contrast, for wireless sensor networks, the systems are wireless, have scarce power, are real-time, utilize sensors and actuators as interfaces, have dynamically changing sets of resources, aggregate behavior is important and location is critical Vampire attacks are not protocol-specific, in that they do not rely on design properties or implementation faults of particular routing protocols, but rather exploit general properties of protocol classes such as link-state, distance-vector, source routing and geographic and beacon routing. Neither do these attacks rely on flooding the network with large amounts of data, but rather try to transmit as little data as possible to achieve the largest energy drain, preventing a rate limiting solution. Since Vampires use protocol-compliant messages, these attacks are very difficult to detect and prevent. The design of the attack model and the mitigation methods are described in the following modules. These attacks are distinct from previously studied DoS, reduction of quality (RoQ), and routing infrastructure attacks as they do not disrupt immediate availability, but rather work over time to entirely disable a network. Vampire attacks are not protocol-specific, in that they do not rely on design properties or implementation faults of particular routing protocols, but rather exploit general properties of protocol classes such as link-state, distance vector, source routing, and geographic and beacon routing. The first challenge in addressing Vampire attacks is defining them—what actions in fact constitute an attack DoS attacks in wired networks are frequently characterized by amplification.

Creating the network scenario

In communication networks, a topology is a usually schematic description of the arrangement of a network, including its nodes and connecting lines. There are two ways of defining network geometry: the physical topology and the logical (or signal) topology.The physical topology of a network is the actual geometric layout of workstations.
Logical (or signal) topology refers to the nature of the paths the signals follow from node to node. The number nodes is going to participate in the simulation is decided. Here we conduct experiments to a group of wireless nodes in a network that operates on the Dynamic Source Routing (DSR) protocol. We hence use only a logical topology as it is wireless environment.

Protocols and assumptions

we consider the effect of Vampire attacks on link-state, distance-vector, source routing, and geographic and beacon routing protocols, as well as a logical ID-based sensor network routing protocol. All routing protocols employ at least one topology discovery period, since ad hoc deployment implies no prior position knowledge.

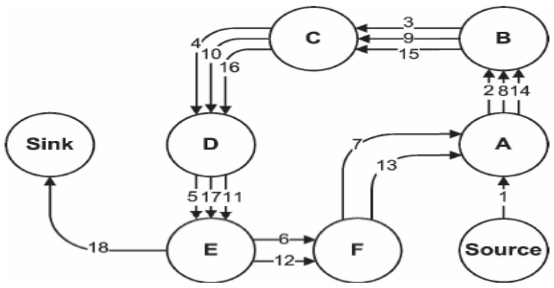


Fig.2 carousel attack

DYNAMIC SOURCE ROUTING

Dynamic Source Routing (DSR) is a routing protocol for wireless mesh networks. It is similar to AODV in that it forms a route on-demand when a transmitting computer requests one. However, it uses source routing instead of relying on the routing table at each intermediate device. Determining source routes requires accumulating the address of each device between the source and destination during route discovery. The accumulated path information is cached by nodes processing the route discovery packets. The learned paths are used to route packets. To accomplish source routing, the routed packets contain the address of each device the packet will traverse. This may result in high overhead for long paths or large addresses, like IPv6. To avoid using source routing, DSR optionally defines a flow id option that allows packets to be forwarded on a hop-by-hop basis. This protocol is truly based on source routing whereby all the routing information is maintained (continually updated) at mobile nodes. It has only two major phases, which are Route Discovery and Route Maintenance. Route Reply would only be generated if the message has reached the intended destination node (route record which is initially contained in Route Request would be inserted into the Route Reply).To return the Route Reply, the destination node must have a route to the source node. If the route is in the Destination Node's route cache, the route would be used. Otherwise, the node will reverse the route based on the route record in the Route Request message header (this requires that all links are symmetric). In the event of fatal transmission, the Route Maintenance Phase is initiated whereby the Route Error packets are generated at a node. The erroneous hop will be removed from the node's route cache; all routes containing the hop are truncated at that point. Again, the Route Discovery Phase is initiated to determine the most viable route. For information on other similar protocols, see the ad hoc routing protocol list. Dynamic source routing protocol (DSR) is an on-demand protocol designed to restrict the bandwidth consumed by control packets in ad hoc wireless networks by eliminating the periodic table-update messages required in the table-driven approach. The major difference between this and the other on-demand routing protocols is that it is beacon-less and hence does not require periodic hello packet (beacon) transmissions, which are used by a node to inform its neighbors of its presence. The basic approach of this protocol (and all other on-demand routing protocols) during the route construction phase is to establish a route by flooding Route Request packets in the network. The destination node, on receiving a Route Request packet, responds by sending a Route Reply packet back to the source, which carries the route traversed by the Route Request packet received .Consider a source node that does not have a route to the destination. When it has data packets to be sent to that destination, it initiates a Route Request packet. This Route Request is flooded throughout the network. Each node, upon receiving a Route Request packet, rebroadcasts the packet to its neighbors if it has not forwarded it already, provided that the node is not the destination node and that the packet’s time to live (TTL) counter has not been exceeded. Each Route Request carries a sequence number generated by the source node and the path it has traversed. A node, upon receiving a Route Request packet, checks the sequence number on the packet before forwarding it. The packet is forwarded only if it is not a duplicate Route Request. The sequence number on the packet is used to prevent loop formations and to avoid multiple transmissions of the same Route Request by an intermediate node that receives it through multiple paths. Thus, all nodes except the destination forward a Route Request packet during the route construction phase. A destination node, after receiving the first Route Request packet, replies to the source node through the reverse path the Route Request packet had traversed. Nodes can also learn about the neighboring routes traversed by data packets if operated in the promiscuous mode (the mode of operation in which a node can receive the packets that are neither broadcast nor addressed to itself). This route cache is also used during the route construction phase.

ATTACKS ON STATELESS PROTOCOLS

Here, we present simple but previously neglected attacks on source routing protocols, such as DSR . In these systems, the source node specifies the entire route to a destination within the packet header, so intermediaries do not make independent forwarding decisions, relying rather on a route specified by the source. To forward a message, the intermediate node finds itself in the route (specified in the packet header) and transmits the message to the next hop.

The burden is on the source to ensure that the route is valid at the time of sending, and that every node in the route is a physical neighbor of the previous route hop. This approach has the advantage of requiring very little forwarding logic at intermediate nodes, and allows for entire routes to be sender authenticated using digital signatures, as in Ariadne . We evaluated both the carousel and stretch attacks (Fig. 1a) in a randomly generated 30-node topology and a single randomly selected malicious DSR agent, using the ns- 2 network simulator . Energy usage is measured for the minimum number of packets required to deliver a single message, so sending more messages increases the strength of the attack linearly until bandwidth saturation.1 We independently computed resource utilization of honest and malicious nodes and found that malicious nodes did not use a disproportionate amount of energy in carrying out the attack. In other words, malicious nodes are not driving down the cumulative energy of the network purely by their own use of energy. Nevertheless, malicious node energy consumption data are omitted for clarity. The attacks are carried out by a randomly selected adversary using the least intelligent attack strategy to obtain average expected damage estimates. More intelligent adversaries using more information about the network would be able to increase the

1. Energy is debited from nodes only for packet transmission, and not for reception or processing, so the number of neighbours of the malicious node is not a confounding variable. strength of their attack by selecting destinations designed to maximize energy usage. Per-node energy usage under both attacks is shown in

Fig. 2. As expected, the carousel attack causes excessive energy usage for a few nodes, since only nodes along a shorter path are affected. In contrast, the stretch attack shows more uniform energy consumption for all nodes in the network, since it lengthens the route, causing more nodes to process the packet. While both attacks significantly network-wide energy usage, individual nodes are also noticeably affected, with some losing almost 10 percent of their total energy reserve per message. Fig. 3a diagrams the energy usage when node 0 sends a single packet to node 19 in an example network topology with only honest nodes. Black arrows denote the path of the packet.

Carousel attack

In this attack, an adversary sends a packet with a route composed as a series of loops, such that the same node appears in the route many times. This strategy can be used to increase the route length beyond the number of nodes in the network, only limited by the number of allowed entries in the source route.2. In Fig. 3, malicious node 0 carries out a carousel attack, sending a single message to node 19 (which does not have to be malicious). Note the drastic increase in energy usage along the original path.3 Assuming the adversary limits the transmission rate to avoid saturating the network, the theoretical limit of this attack is an energy usage increase factor of O(λ), where λ is the maximum route length.

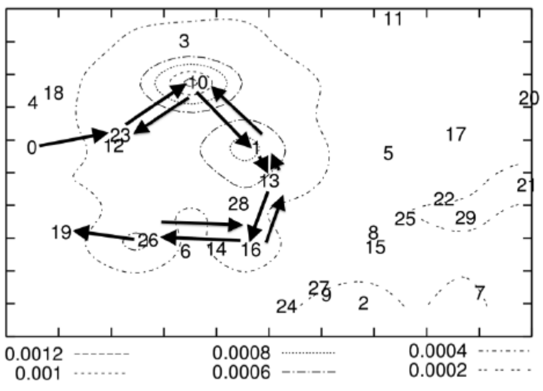


Fig.3 Carousel attack

Overall energy consumption increases by up to a factor of 3.96 per message. On average, a randomly located carousel attacker in our example topology can increase network energy consumption by a factor of 1.48 ± 0.99. The reason for this large standard deviation is that the attack does not always increase energy usage—the length of the adversarial path is a multiple of the honest path, which is in turn, affected by the position of the adversary in relation to the destination, so the adversary's position is important to the success of this attack.

Stretch attack

Another attack in the same vein is the stretch attack, where a malicious node constructs artificially long source routes, causing packets to traverse a larger than optimal number of nodes. An honest source would select the route Source→F→E→Sink, affecting four nodes including itself, but the malicious node selects a longer route, affecting all nodes in the network. These routes cause nodes that do not lie along the honest route to consume energy by forwarding packets they would not receive in honest scenario. The outcome becomes clearer when we examine Fig. 4 and compare to the carousel attack. While the latter uses energy at the nodes who were already in the honest path, the former extends the consumed energy "equivalence lines" to a wider section of the network. Energy usage is less localized around the original path, but more total energy is consumed.

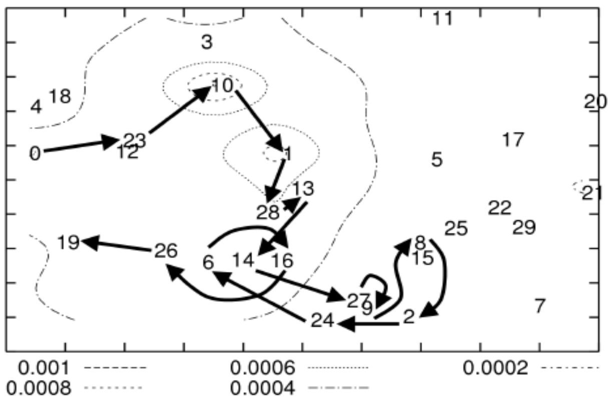


Fig. 4 Stretch attack

The theoretical limit of the stretch attack is a packet that traverses every network node, causing an energy usage increase of factor O(min(N,λ)), where N is the number of nodes in the network and λ is the maximum path length allowed. This attack is potentially less damaging per packet than the carousel attack, as the number of hops per packet is bounded by the number of network nodes. However, adversaries can combine carousel and stretch attacks to keep the packet in the network longer: the resulting "stretched cycle" could be traversed repeatedly in a loop. Therefore, even if stretch attack protection is not used, route loops should still be detected and removed to prevent the combined attack. In our example topology, we see an increase in energy usage by as much as a factor of 10.5 per message over the honest scenario, with an average increase in energy consumption of 2:67 _ 2:49. As with the carousel attack, the reason for the large standard deviation is that the position of the adversarial node affects the strength of the attack. Not all routes can be significantly lengthened, depending on the location of the adversary. Unlike the carousel attack, where the relative positions of the source and sink are important, the stretch attack can achieve the same effectiveness independent of the attacker's network position relative to the destination, so the worst case effect is far more likely to occur.

MITIGATION METHODS

The carousel attack can be prevented entirely by having forwarding nodes check source routes for loops. While this adds extra forwarding logic and thus more overhead, we can expect the gain to be worthwhile in malicious environments. The ns-2 DSR protocol does implement loop detection, but confusingly does not use it to check routes in forwarded packets.5 When a loop is detected, the source route could be corrected and the packet sent on, but one of the attractive features of source routing is that the route can itself be signed by the source. Therefore, it is better to simply drop the packet, especially considering that the sending node is likely malicious (honest nodes should not introduce loops). An alternate solution is to alter how intermediate nodes process the source route. To forward a message, a node must determine the next hop by locating itself in the source route. If a node searches for itself from the destination backward instead from the source forward, any loop that includes the current node will be automatically truncated (the last instance of the local node will be found in the source route rather than the first). No extra processing is required for this defense, since a node must perform this check anyway—we only alter the way the check is done. The stretch attack is more challenging to prevent. Its success rests on the forwarding node not checking for optimality of the route. If we call the no-optimization case "strict" source routing, since the route is followed exactly as specified in the header, we can define loose source routing, where intermediate nodes may replace part or all of the route in the packet header if they know of a better route to the destination. This makes it necessary for nodes to discover and cache optimal routes to at least some fraction of other nodes, partially defeating the as-needed discovery advantage. Moreover, caching must be done carefully lest a maliciously suboptimal route be introduced.

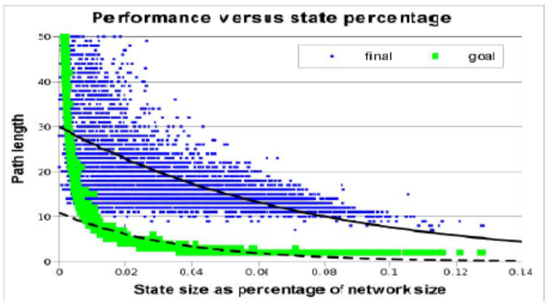


Fig. 5. Loose source routing performance compared to optimal, in a network with diameter slightly above 10. The dashed trend line represents expected path length when nodes store $\log N$ local state, and the solid trend line shows actual observed performance.

We simulated the loose source routing defense using random-length suboptimal paths in randomly generated network topologies of up to 1,000,000 nodes, with diameter 10-14. Results (Fig. 5) demonstrate that the amount of node-local storage required toachieve reasonable levels of mitigation approaches global topology knowledge, defeating the purpose of using source routing. The dashed trend line represents the expected path length of rerouted packets if each node stores logN network paths, where N is the number of network nodes, while the solid trend line represents the majority of actual network paths in a loose source-routing setup. The number of nodes traversed by loose source routed packets is suboptimal by at least a factor of 10, with some routes approaching a factor of 50. Only a few messages encountered a node with a better path to the destination than the originally assigned long source route. Therefore we conclude that loose source routing is worse than keeping global state at every node. Alternatively, we can bound the damage of carousel and stretch attackers by limiting the allowed source route length based on the expected maximum path length in the network, but we would need a way to determine the network diameter.6 While there are suitable algorithms there has been very little work on whether they could yield accurate results in the presence of adversaries. If the number of nodes is known ahead of time, graph-theoretic techniques can be used to estimate the diameter. Rate limiting may initially seem to be a good defense, but upon closer examination we see it is not ideal. It limits malicious sending rate, potentially increasing network lifetime, but that increase becomes the maximum expected lifetime, since adversaries will transmit at the maximum allowed rate. Moreover, sending rate is already limited by the size of nodes' receive queues in rate-unlimited networks (as seen in the 10,000 message scenario in Fig. 4a). Rate limiting also potentially punishes honest nodes that may transmit large amounts of time-critical (bursty) data, but will send little data over the network lifetime.

Malicious discovery attack:

Another attack on all previously mentioned routing protocols (including stateful and stateless) is spurious route discovery. In most protocols, every node will forward route discovery packets (and sometimes route responses as well), meaning it is possible to initiate a flood by sending a single message. Systems that perform as-needed route discovery, such as AODV and DSR, are particularly vulnerable, since nodes may legitimately initiate discovery at any time, not just during a topology change. A malicious node has a number of ways to induce a perceived topology change: it may simply falsely claim that a link is down, or claim a new link to a nonexistent node. Security measures, such as those proposed by Raffo et al. in ay be sufficient to alleviate this particular problem. Further, two cooperating malicious nodes may claim the link between them is down. However, nearby nodes might be able to monitor communication to detect link failure (using some kind of neighbourhood update scheme). Still, short route failures can be safely ignored in networks of sufficient density. More serious attacks become possible when nodes claim that a long distance route has changed. This attack is trivial in open networks with unauthenticated routes, since a single node can emulate multiple nodes in neighbour relationships , or falsely claim nodes as neighbours. Therefore, let us assume closed (Sybil-resistant) networks where link states are authenticated, similar to route authentication in Ariadne or path-vector signatures. Now our adversary must present an actually changed route in order to execute the attack. To do this, two cooperating adversaries communicating through a wormhole could repeatedly announce and withdraw routes that use this wormhole, causing a theoretical energy usage increase of a factor of O(N) per packet. Adding more malicious nodes to the mix increases the number of possible route announce/withdrawal pairs. Packet Leashes cannot prevent this attack, with the reasoning being similar to the directional antenna attack—since the originators are themselves malicious, they would forward messages through the wormhole, and return only seemingly valid (and functional) routes in response to discovery. This problem is similar to route flapping in BGP , but while Internet paths are relatively stable paths change frequently in wireless ad hoc networks, where nodes may move in and out of each other’s range, or suffer intermittent environmental effects. Since there may be no stable routes in WSNs (hence the need for ad hoc protocols), this solution would not be applicable.

PLGP in the Presence of Vampires

In PLGP, forwarding nodes do not know what path a packet took, allowing adversaries to divert packets to any part of the network, even if that area is logically further away from the destination than the malicious node. This makes PLGP vulnerable to Vampire attacks. Consider for instance the now-familiar directional antenna attack: a receiving honest node may be farther away from the packet destination than the malicious forwarding node, but the honest node has no way to tell that the packet it just received is moving away from the destination; the only information available to the honest node is its own address and the packet destination address, but not the address of the previous hop (who can lie). Thus, the Vampire can move a packet away from its destination without being detected. This packet will traverse at most logN logical hops, $O(\sqrt{2^i})$ physical hops at the ith logical hop, giving us a theoretical maximum energy increase of O(d), where d is the network diameter and N the number of network nodes. The situation is worse if the packet returns to the Vampire in the process of being forwarded—it can now be rerouted again, causing something similar to the carousel attack. Recall that the damage from the carousel attack is bounded by the maximum length of the source route, but in PLGP the adversary faces no such limitation, so the packet can cycle indefinitely. Nodes may sacrifice some local storage to retain a record of recent packets to prevent this attack from being carried out repeatedly with the same packet.

```
Function forward_packet(p)
    s ← extract_source_address(p);
    c ← closest_next_node(s);
    if is_neighbor(c) then forward(p, c);
    else
        r ← next_hop_to_non_neighbor(c);
        forward(p, r);
```

Random direction vectors, as suggested in PLGP, would likewise alleviate the problem of indefinite cycles by avoiding the same malicious node during the subsequent forwarding round.

Provable security against vampire attacks

Here, we modify the forwarding phase of PLGP to provably avoid the above-mentioned attacks. First we introduce the no-backtracking property, satisfied for a given packet if and only if it consistently makes progress toward its destination in the logical network address space. More formally:

Definition 1. No-backtracking is satisfied if every packet p traverses the same number of hops whether or not an adversary is present in the network. (Maliciously induced route stretch is bounded to a factor of 1.)

This does not imply that every packet in the network must travel the same number of hops regardless of source or destination, but rather that a packet sent to node D by a malicious node at location L will traverse the same number of hops as a packet sent to D by a node at location L that is honest. If we think of this in terms of protocol execution traces, no-backtracking implies that for each packet in the trace, the number of intermediate honest nodes traversed by the packet between source and destination is independent of the actions of malicious nodes. Equivalently, traces that include malicious nodes should show the same network wide energy utilization by honest nodes as traces of a network with no malicious actors. The only notable exceptions are when adversaries drop or mangle packets en route, but since we are only concerned with packets initiated by adversaries, we can safely ignore this situation: “premangled” packets achieve the same result—they will be dropped by an honest intermediary or destination.

No-backtracking implies Vampire resistance. It is not immediately obvious why no-backtracking prevents Vampire attacks in the forwarding phase. Recall the reason for the success of the stretch attack: intermediate nodes in a source route cannot check whether the source-defined route is optimal, or even that it makes progress toward the destination. When nodes make independent routing decisions such as in link-state, distance-vector, coordinate- based, or beacon-based protocols, packets cannot contain maliciously composed routes. This already means the adversary cannot perform carousel or stretch attacks— no node may unilaterally specify a suboptimal path through the network. However, a sufficiently clever adversary may still influence packet progress. We can prevent this interference by independently checking on packet progress: if nodes keep track of route “cost” or metric and, when forwarding a packet, communicate the local cost to the next hop, that next hop can verify that the remaining route cost is lower than before, and therefore the packet is making progress toward its destination. (Otherwise we suspect malicious intervention and drop the packet.) If we can guarantee that a packet is closer to its destination with every hop, we can bound the potential damage from an attacker as a function of network size. PLGP does not satisfy no-backtracking. In nonsource routing protocols, routes are dynamically composed of forwarding decisions made independently by each node. PLGP differs from other protocols in that packets paths are further bounded by a tree, forwarding packets along the shortest route through the tree that is allowed by the physical topology. In other words, packet paths are constrained both by physical neighbor relationships and the routing tree. Since the tree implicitly mirrors the topology (two nodes have the same parent if and only if they are physical neighbors, and two nodes sharing an ancestor have a network path to each other), and since every node holds an identical copy of the address tree, every node can verify the optimal next logical hop.

```
Function secure_forward_packet (p)
  s ← extract_source_address (p);
  a ← extract_attestation (p);
  if (not verify_source_sig (p)) or
  (empty (a) and not is_neighbor (s)) or
  (not saowf_verify (a)) then
      return ;                                   /* drop (p) */
  foreach node in a do
      prevnode ← node;
      if (not are_neighbors (node, prevnode)) or
      (not making_progress (prevnode, node)) then
          return ;                               /* drop (p) */

  c ← closest_next_node (s);
  p′ ← saowf_append (p);
  if is_neighbor (c) then forward (p′, c);
  else forward (p′, next_hop_to_non_neighbor (c));
```

However, this is not sufficient for no-backtracking to hold, since nodes cannot be certain of the path previously traversed by a packet. Communicating a local view of route cost is not as easy as it seems, since adversaries can always lie about their local metric, and so PLGP is still vulnerable to directional antenna/wormhole attacks, which allow adversaries to divert packets to any part of the network.

To preserve no-backtracking, we add a verifiable path history to every PLGP packet, similar to route authentications in Ariadne and path-vector signatures. The resulting protocol, PLGP with attestations (PLGPa) uses this packet history together with PLGP's tree routing structure so every node can securely verify progress, preventing any significant adversarial influence on the path taken by any packet which traverses at least one honest node. Whenever node n forwards packet p, it this by attaching a nonreplayable attestation (signature). These signatures form a chain attached to every packet, allowing any node receiving it to validate its path. Every forwarding node verifies the attestation chain to ensure that the packet has never traveled away from its destination in the logical address space. See Function secure_forward_packet for the modified protocol.

PLGPa satisfies no-backtracking. To show that our modified protocol preserves the no-backtracking property, we define a network as a collection of nodes, a topology, connectivity properties, and node identities, borrowing the model used by Poturalski]. Honest nodes can broadcast and receive messages, while malicious nodes can also use directional antennas to transmit to (or receive from) any node in the network without being overheard by any other node. Honest nodes can compose, forward, accept, or drop messages, and malicious nodes can also arbitrarily transform them. Our adversary is assumed to control m nodes in an N-node network (with their corresponding identity certificates and other secret cryptographic material) and has perfect knowledge of the network topology. Finally, the adversary cannot affect connectivity between any two honest nodes.

Since all messages are signed by their originator, messages from honest nodes cannot be arbitrarily modified by malicious nodes wishing to remain undetected. Rather, the adversary can only alter packet fields that are changed en route (and so are not authenticated), so only the route attestation field can be altered, shortened, or removed entirely. To prevent truncation, which would allow Vampires to hide the fact that they are moving a packet away from its destination, we use Saxena and Soh's one-way signature chain construction , which allow nodes to add links to an existing signature chain, but not remove links, making attestations append only.

DSDV:

In DSDV the Packets are transmitted by using "Route Table".

To Maintain Route Table

1..Each node broadcasts its Route Table Entries Periodically

And Immediately

2.When a node receive new routing information

1.More recent Sequence Number is used.

2.With the same Sequence Number, the smallest metric will be used

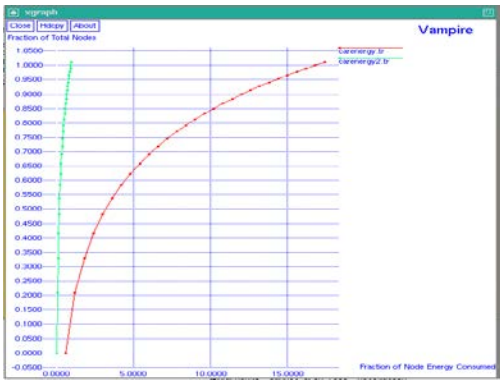
3.Metric+1, then broadcast its route information.

The route information broadcast by each node contain its new Sequence Number, each route, the Destination's address,the number of hops required to reach the destination.

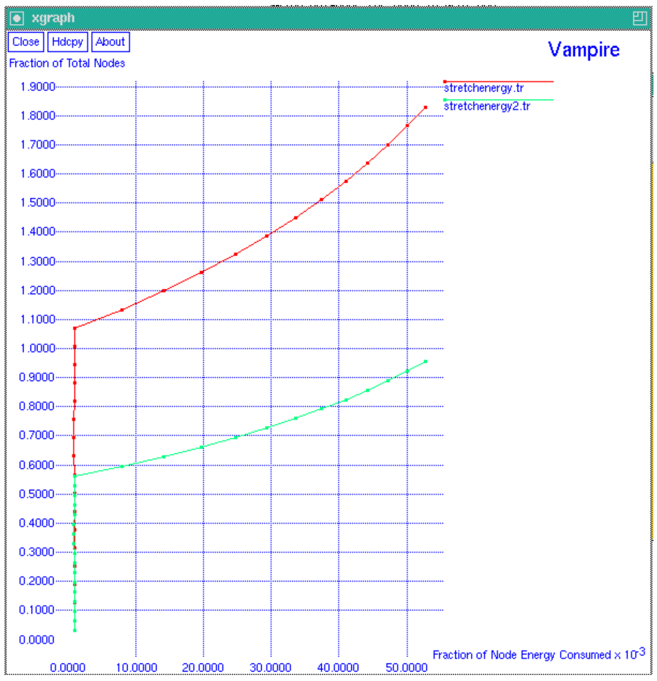
In these paper we will show later that a single Vampire may attack every network node simultaneously, meaning that continuous recharging does not help unless Vampires are more resource constrained than honest nodes. Dual-cycle networks (with mandatory sleep and awake periods) are equally vulnerable to Vampires during active duty as long as the Vampire's cycle switching is in sync with other nodes. Vampire attacks may be weakened by using groups of nodes with staggered cycles: only active-duty nodes are vulnerable while the Vampire is active; nodes are safe while the Vampire sleeps. However, this defense is only effective when duty cycle groups outnumber Vampires, since it only takes one Vampire per group to carry out the attack. Also we present a series of increasingly damaging Vampire attacks, evaluate the vulnerability of several example protocols, and suggest how to improve resilience.

Carousal Energy

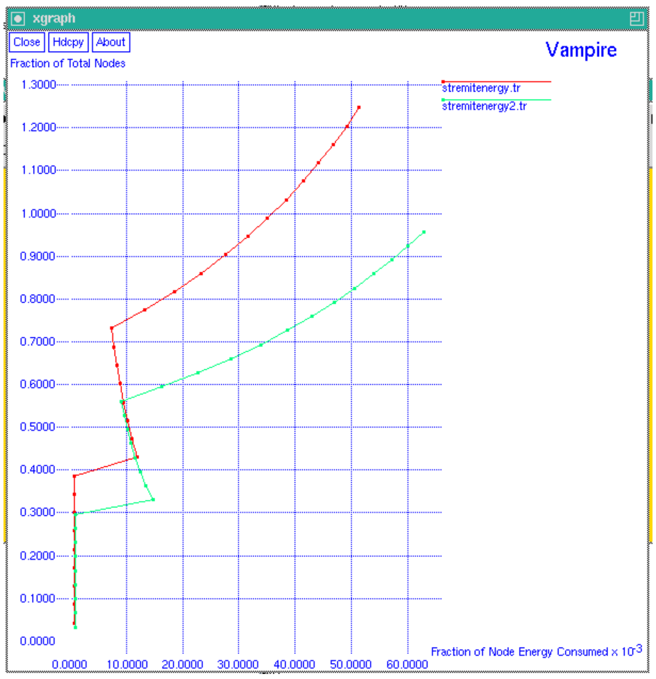
Energy consumption is high in DSR. Energy consumption is low in DSDV.so that we are increasing the node's lifetime.



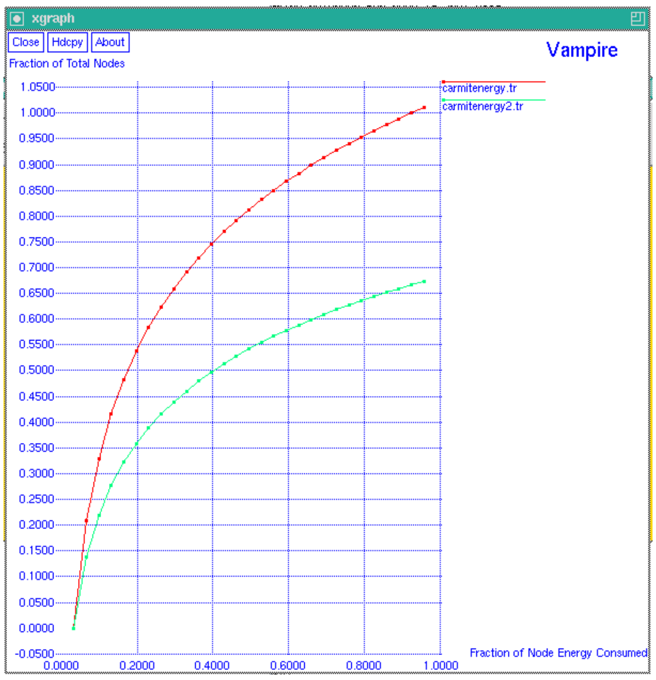
Stretch Attack



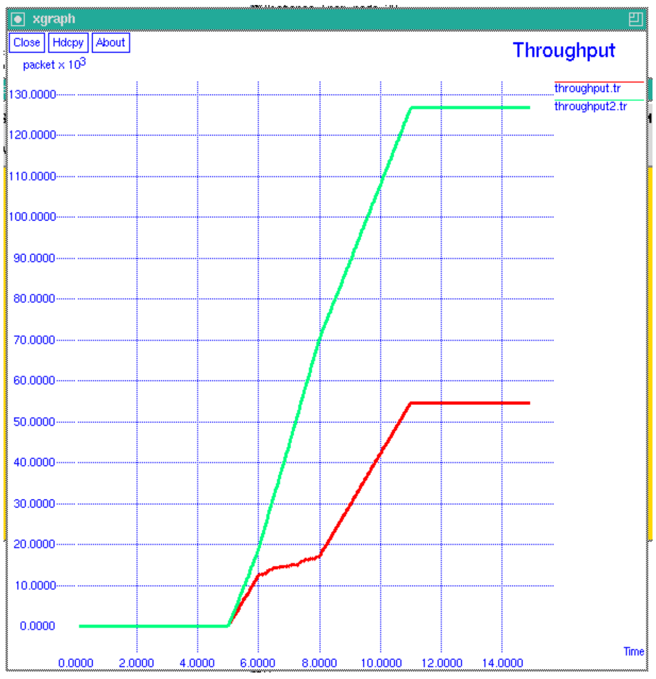
Stretch Mitigation



Carousal Mitigation



Throughput



CONCLUSION

Thus the vampire attacks in the wireless sensor networks have been studied. An attack model of the carousel and stretch attacks have been shown in ns2 and simulated for energy performance analysis. A modification in the PLGP method to mitigate the vampire attacks in the WSNs have been proposed and proved to be efficient in the network. Even though it is not possible to completely eliminate the vampire attacks, the possible mitigation methods to mitigate both the attacks were performed. The energy graphs stand as proof of concept for the proposed work.

The use of DSDV protocol reduces overhead in the network and helps in mitigating vampire attack in a better way. This has been used as a motive to implement the same in the future work. The DSDV protocol will be used to simulate and analyse both the attack models to perform mitigation methods and see how well the attacks are mitigated.

ACKNOWLEDGEMENT

I like to express heart full thanks to A.sivashakthivel, assistant Professor, Department of ECE, to spend his valuable time to do this project and also I want to thank to my college and my friends for their support.

REFERENCES

[1] “The Network Simulator - ns-2,” http://www.isi.edu/nsnam/ns, 2012.

[2] I. Aad, J.-P. Hubaux, and E.W. Knightly, “Denial of Service Resilience in Ad Hoc Networks,” Proc. ACM MobiCom, 2004.

[3] G. Acs, L. Buttyan, and I. Vajda, “Provably Secure On-Demand Source Routing in Mobile Ad Hoc Networks,” IEEE Trans. Mobile
Computing, vol. 5, no. 11, pp. 1533-1546, Nov. 2006.

[4] T. Aura, “Dos-Resistant Authentication with Client Puzzles,” Proc. Int’l Workshop Security Protocols, 2001.

[5] J. Bellardo and S. Savage, “802.11 Denial-of-Service Attacks: Real Vulnerabilities and Practical Solutions,” Proc. 12th Conf. USENIX Security, 2003.

[6] D. Bernstein and P. Schwabe, “New AES Software Speed Records,” Proc. Ninth Int’l Conf. Cryptology in India: Progress in Cryptology (INDOCRYPT), 2008.

[7] D.J. Bernstein, “Syn Cookies,” http://cr.yp.to/syncookies.html, 1996.

[8] I.F. Blaked, G. Seroussi, and N.P. Smart, Elliptic Curves in Cryptography, vol. 265. Cambridge Univ. , 1999.

[9] J.W. Bos, D.A. Osvik, and D. Stefan, “Fast Implementations of AES on Various Platforms,” Cryptology ePrint Archive, Report 2009/ 501, http://eprint.iacr.org, 2009.

[10] H. Chan and A. Perrig, “Security and Privacy in Sensor Networks,” Computer, vol. 36, no. 10, pp. 103-105, Oct. 2003.