World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:3, No:4, 2009

Improving Packet Latency of Video Sensor Networks

Arijit Ghosh and Tony Givargis

Abstract—Video sensor networks operate on stringent requirements of latency. Packets have a deadline within which they have to be delivered. Violation of the deadline causes a packet to be treated as lost and the loss of packets ultimately affects the quality of the application. Network latency is typically a function of many interacting components. In this paper, we propose ways of reducing the forwarding latency of a packet at intermediate nodes. The forwarding latency is caused by a combination of processing delay and queueing delay. The former is incurred in order to determine the next hop in dynamic routing. We show that unless link failures in a very specific and unlikely pattern, a vast majority of these lookups are redundant. To counter this we propose source routing as the routing strategy. However, source routing suffers from issues related to scalability and being impervious to network dynamics. We propose solutions to counter these and show that source routing is definitely a viable option in practical sized video networks. We also propose a fast and fair packet scheduling algorithm that reduces queueing delay at the nodes. We support our claims through extensive simulation on realistic topologies with practical traffic loads and failure patterns.

Keywords—Sensor networks, Packet latency, Network design, Network performance.

I. Introduction

SENSOR networks are becoming increasingly popular as non-intrusive surveillance systems. Resource-rich wired sensor networks, or example Ethernet and ATM-based video surveillance network (VSN) using intelligent cameras, are a prime example. The Dallas-Fort Worth International Airport has deployed a VSN [1] that produces high-resolution, full-motion, broadcast-quality color images and audio in place of the current systems that provide grainy black and white, or poor quality color video images. The city of New Orleans uses a similar system at high-traffic spots and high-crime neighborhoods [2]. In early 2004, the Technology Office worked with the New Orleans Police Department. Using detailed crime maps drawn with the city's COMSTAT crime analysis and management tools, they selected areas with a large number of murders, robberies, vehicle thefts and drug trafficking. As a result of this, the First District recorded 57% fewer murders and 30% fewer car thefts within a year of their deployment.

To be useful, VSN applications must incur minimum delay from when the event occurred to when the incident was reported. Consider a terrorist attack in a shopping mall equipped with a VSN. In order for the security team to react effectively, the VSN should be able to quickly detect the event, identify the suspects, and provide real-time tracking of their movement. All aspects of the system - application, operating system, network and hardware - affect the application latency. In this paper, we focus on network latency which we define as the *one-way* delay incurred by a packet from the time when it is transmitted by the source to the time when it is received by the destination. The packet latency is the sum of the propagation and the transmission delays at each link along its route and the forwarding delay at each intermediate node. We consider the forwarding delay from after the reception of the packet to just before its transmission. Thus, the forwarding delay is a combination of the queueing delay and processing delay. The latter is made up of such activities as routing lookup, header modification, buffer copying and so on. In this paper, we propose ways of reducing the forwarding delay and hence reduce the packet latency.

Queueing delay occurs when a physical link has to be shared among multiple packets. One way to reduce this delay is to reduce the number of packets in the network. This approach is known as traffic shaping and typically occurs before the packet enters the network. The other approach, which is more relevant to this paper, is to reduce the processing time of packet scheduling algorithms. Packet schedulers need to be fast and fair. There are usually two categories of packet schedulers that can be found commonly in literature: those that emphasize speed, for example round-robin schedulers, and those that emphasize fairness, for example timestamp schedulers. In this paper we propose a scheduling algorithm that attempts to bridge the gap between the two. It has very good fairness characteristics, is extremely simple making it amenable to a hardware implementation and provides the latency bound on a single packet.

The processing delay associated at intermediate nodes is incurred mainly in determining the next hop (in case of dynamic routing). If source routing is used instead, this delay can be potentially eliminated which could have a big impact on packet latencies. Source routing is inherently not scalable. This is because each node has to maintain the complete path to all nodes in the network. Further, inclusion of the path increases the size of the header which reduces the goodput of the system. In this paper, we analyze a typical VSN and show that source routing *IS* feasible in practical deployments. The number of hops in a random topology grow at a much slower rate of $O(\log N)$. If the topology be built to reflect a scale-free network, then the above can be reduced to $O(\log \log N)$. We propose a topology encoding scheme that places modest memory requirements at the nodes and reduces the packet header overhead. A source route usually does not change once specified. We break this assumption and propose a lazy correction scheme.

A. Ghosh and T. Givargis are with University of California, Irvine, CA 92697, USA.

Manuscript received April 1, 2009; revised May 1, 2009.

It is interesting to note that making source routing practical can have other positive impacts. A known way of reducing congestion, and consequently queueing delay, is to use QoS routing. In this method, packets are routed along the *fastest* and not necessarily the shortest path. Although attractive, QoS routing is computationally expensive to be implemented as a dynamic protocol but becomes much more feasible with source routing.

When source routing is used, then all an intermediate node has to do is to forward the packet from the input to the output interface and completely bypass all other kinds of processing. Recognizing this, we propose a short-circuiting scheme that operates at the link layer. We propose a simple hardware implementation which should make the forwarding operate at wire speeds.

To summarize, in this paper, we look at specific techniques to reduce the forwarding delay at each intermediate node. In particular, we make the following contributions.

- We show that source routing is an attractive alternative that can be made practical in VSNs (Section 4). We propose a topology encoding for the same. We combine source routing with *lazy correction* to react to network dynamics (Section 5).
- We propose a a link-layer *short circuiting* technique to improve packet switching (Section 6).
- We present an approximation of the WFQ algorithm that is amenable to hardware implementation to improve the packet scheduling latency (Section 7).
- We present thorough evaluation of our system by simulating realistic network topologies using practical traffic types and failure models (Section Section 8).

II. Previous Work

Techniques to improve packet latency have long been a subject of intensive research in the networking community. Many approaches targeting different parts of the networking subsystem have been proposed. The speed of physical links have improved dramatically by the deployment of fiber optic networks and Wavelength Division multiplexing (WDM). Processing speed at intermediate nodes have been improved by using smart network processors that use different techniques for fast route lookup. Several smart packet scheduling algorithms [3], [4], [5], [6], [7], [8] provide efficient queueing algorithms that aims to reduce the waiting of a packet in a queue.

There are many service models and mechanisms to reduce packet latency. The Integrated Services [9] model is characterized by resource reservation. For real-time applications, before data are transmitted, the applications must first set up paths and reserve resources. RSVP is a signaling protocol for setting up paths and reserving resources and is based on traffic characteristics. In Differentiated Services [10], packets are marked differently to create several packet classes. Packets in different classes receive different services. MPLS [11] is a forwarding scheme. Packets are assigned labels at the ingress of a MPLS-capable domain. Subsequent classification, forwarding, and services for the packets are based on the labels. Traffic Engineering is the process of arranging how traffic flows through the network. The idea is to reduce congestion in the network by routing packets along alternate routes. Constraint Based Routing is to find routes that are subject to some constraints such as bandwidth or delay requirement. A thorough discussion of all the relevant approaches can be found in [12]. An approach to provide guaranteed delay is to use a network with Weighted Fair Queueing (WFQ) switches and a data flow that is leaky bucket constrained [13]. However, this requires that traffic be shaped at the source to conform to a specific characteristic.

There is a significant amount of prior work in finding scheduling disciplines that provide delay and fairness guarantees. Generalized Processor Sharing [13] (also called Fluid Fair Queuing) is considered the ideal scheduling discipline and acts as a benchmark for other scheduling disciplines. Practical scheduling disciplines can be broadly classified as either timestamp schedulers or round-robin schedulers.

Timestamp schedulers [3], [4], [6], [7] try to emulate the operation of GPS by computing a timestamp for each packet. Packets are then transmitted in increasing order of their timestamps. Although timestamp schedulers have good delay properties, they suffer from a sorting bottleneck that results in a time complexity of $O(\log F)$ per packet. Any scheduler that has a complexity of below $O(\log F)$ must incur a GPS-relative delay proportional to F, the number of flows [14].

Round-robin schedulers [5], [8], [15], [16] are the other broad class of work-conserving schedulers. These schedulers typically assign time slots to flows in some sort of round-robin fashion. By eliminating the sorting bottleneck associated with timestamp schedulers, they achieve an O(1) time packet processing complexity. As a result, they tend to have poor delay bounds and output burstiness.

Stratified round robin [17] is an attempt to bridge the gap between the simplicity of round robin schedulers and the fairness of timestamp schedulers. SRR groups flows of roughly similar bandwidth requirements into a single flow class. Within a flow class, a weighted round robin scheme is employed. However, deadline based scheduling is employed over flow classes. Although the number of slows to be sorted is reduced by this method, it still is partially dependent on the efficiency of the sorting algorithm.

In contrast to the above approaches, we present a scheduling algorithm that has low complexity, exhibits good fairness and provided a bounded delay for a single packet that is independent of the number of flows.

Source routing (SR) is a very old technique that has been used extensively in both wired and wireless networks including sensor networks [18]. Although simple, SR has not really been popular in wide area networks because it doesn't scale well with the network size. In SR, the entire path that the packet travels is included in it. This implies that a node has to maintain a route to every other node. This imposes an $O(N)$ routing table overhead. The typical approach to solving this problem is by introduction of hierarchies [19]. In this paper, we also propose a topological solution but is different from existing hierarchical solutions. The second problem of scalability comes from the increased size of packet headers due

TABLE II
PROPERTIES OF TOPOLOGIES

Topology	Nodes	Links	Max. Hops	Max. out degree
G0	4000	50000	15	1250
G1	16	24	4	4
G2	19	30	5	5
G3	14	21	4	4

to the inclusion of the path. A solution to this was proposed in [20]. In this, the direction of the neighbors are encoded in the source path. We expand on this idea and make it more general so that it can be applied to topologies with arbitrary number of neighbors.

III. SIMULATOR

We use a two level event-driven simulation. At the session level, the simulator generates flows at each source node. Flows arrive according to a Poisson distribution. We consider flows sending either audio or video data. The flow specifications are shown in Table 1.

At the packet level, it manages the lifetime of a packet. Specifically, it has routing, queueing, failure and delay modules. The routing module simulates a link-state based shortest path algorithm, much like Open Shortest Path First (OSPF). The queueing strategy used is Approximate Fair Queueing which will be discusses in Section 7. We use two failure models. Γ_0 represents a model where links are chosen at random. Γ_1 represents a failure model described in [23]. To decide *when* the failure happens, we picked a random number for the Weibull distribution with parameters α = 0.046 and β = 0.414. We use the power law with slope = -1.35 to decide *where* the failure occurs according to the following. Let n_l be the number of failures occurring in link l over a period of time T. Then the failure probability p_l is proportional to n_l/T. If the links n_1,n_2...,n_L are independent, then the probability that failure happens on link l with probability $p_l/(p_1 + p_2 \ldots + p_L) = n_l/(n_1 + n_2 \ldots + n_L)$. Link states are flooded only when a link fails. We employ a simple flooding strategy as follows. Upon receiving an LSU message, a node updates its database and forwards the information to all its neighbors, except the one from which the update was received. Duplicate messages are discarded. LSU messages are kept in a separate queue and are treated with highest priority by each node.

For our simulations, we use four different topologies, G_0, G_1, G_2 and G_3 (Figure 14). G_0 is a random graph, G_1 is representative of a network within a building while G_2 and G_3 represents a wider area network. Their characteristics are given in Table 2. All links are bidirectional. To reduce the number of variables, we use a constant propagation delay of 10ms for all links. All links are assumed to be 1 Gbps. Transmission speeds for different sized packets are calculated accordingly.

IV. REDUNDANCY

In modern packet switched networks, dynamic routing is the most widely deployed protocol. At each hop, a packet finds the next hop by executing the shortest path algorithm from the current hop to the destination. In practical implementations, each router pre-computes the shortest path to all nodes in the network so that when a packet arrives, determining the next hop amounts to a table lookup. Traversing the different layers of the network stack as well as table lookup at each hop takes up a non-trivial amount of time. An idle Click modular router, for example, takes 30 μs to forward a 64 byte packet [ref:click]. In a completely stable network, all route lookups will be redundant and a considerable amount of time can be saved by using source routing. However, in reality, links fail regularly. To gain an insight on the degree of redundancy in the presence of link failures, we performed the following experiments. We routed 25,000, 50,000 and 100,000 packets between 680,000 randomly chosen pairs of nodes in G_0. Using fault model Γ_0, we injected up to 500 failed links in a manner that 30% of the chosen routes have *at least* one failed link along them. An incredible 93% of the route lookups turned out to be redundant.

To understand the reason behind this, let us consider the case where a single link fails. Let there be a source node A and a destination node B. Let the link between nodes C and D fail. Let P_{old} and P_{new} be the paths between A and B before and after the link fails. If $CD \notin P_{old}$, then $P_{old} = P_{new}$ and all intermediate lookups will be redundant. Let us now consider the case where CD is in P_{old}. Then obviously, $P_{old} \neq P_{new}$. As the link state information propagates from C and D to the rest of the network, all nodes (at any given time) can essentially be divided into two regions: nodes that have received the update and have the newest routes belong to the fresh region; all others belong to the stale region. If A is already in the fresh region, then A will generate P_{new} making all intermediate lookups redundant. Now consider the case when A is in the stale region. A will generate P_{old} as the route of the packet. All intermediate nodes in the stale region will produce the same result as A and hence will cause redundant lookups. As soon as the packet reaches a node in the fresh region, P_{new} will be generated. Lookup in this case is *not* redundant. But from now on until B, again all lookups will be redundant.

To summarize, a route lookup will be different between two nodes belonging to the two different regions but will be the same at all nodes in a given region. This means unless the path of a packet traverses the boundary between the two regions, *all* intermediate route lookups will be redundant.

Let A be n hops away from B and generate r packets/second. Let t and p be the propagation time (between consecutive nodes) and the packet processing time respectively. Let P be the number of packets generated by A before it gets into the fresh region. Thus p is the number of packets that will have exactly one non-redundant route lookup.

$$P = n \times (t + p) \times r$$

Figure 1 shows that P increases linearly with the distance of A from B.

Imagine a packet traversing a n-hop path from P to U as shown in Figure 2. For a route lookup at each hop to be non-redundant, all of the following three conditions must hold.

- The Mean Time To Failure (MTTF) between links must exceed the sum of the average packet processing time

TABLE I
FLOW SPECIFICATIONS

Name	Description	Rate
HDTV	A single channel of High Definition resolution MPEG2 encoded video	20 Mbps [21]
SDTV	A PAL or NTSC-equivalent Standard Definition video	3 Mbps
Stereo	A multichannel DolbyDigital 'AC-3' audio with a maximum 13.1 channels	6 Mbps [22]
Standard Audio	An audio channel encoded with Advanced Audio Coding format	128 Kbps

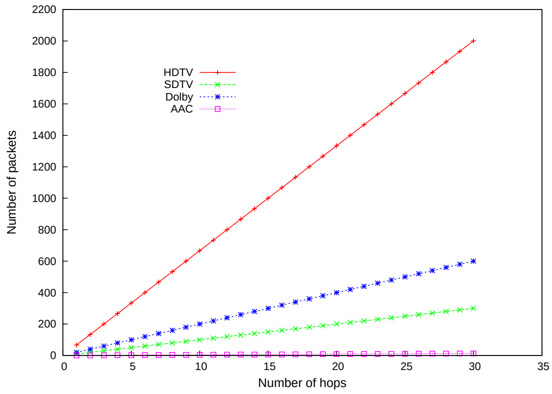


Fig. 1. The number of packets that will incur a redundant lookup before 1 packet has a non-redundant lookup is linearly proportional to the number of hops between the source and the destination.

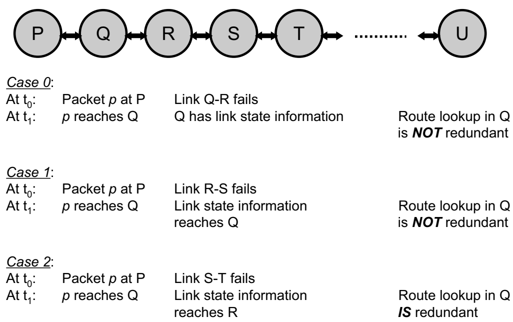


Fig. 2. Redundancy: example topology

and propagation time between two consecutive hops of the packet.

- **All** failures should occur along the path of the packet.
- The currently failed link can be no farther than two hops away from the current hop. Let the link between S and T fail when the packet is at P. Let the processing and propagation time of both the data and link update packets be the same. Then, when the packet reaches Q, the link update will only have reached R. Thus, the lookup at Q will be redundant.

It shows that for routing lookup to be non-redundant at every hop, the failure pattern has to be very specific. Clearly, while possible, there is a low probability for this to happen in real life. Based on this analysis, we infer that source routing is definitely a viable option to be used in VSN.

V. Source Routing with Lazy Correction

Source routing is a simple approach where the sender specifies the complete route of the packet along the network. Although simple and attractive, source routing has a few disadvantages. First, source routing is not scalable. Each node has to maintain the path to all nodes in the network which incurs a space complexity of $O(N)$, Further, the entire route is included in the packet header. This reduces the goodput of the system. Second, once a route is specified, it is not changed. By not taking cognizance of changing network conditions, source routing could cause a packet to traverse a longer path. In the worst case, it might even fail to deliver a packet. Third, source routing presents a security hazard since it allows address spoofing. In this paper we address the issues related to scalability and network dynamics and leave security as part of our future work.

A. Scalability

We address the scalability issues by proposing two techniques. The first is a scheme to encode the topology. The second is to build the sensor network according to a particular topology. Neither of the approaches reduce the storage complexity. But our analysis shows that in practical scenarios, the actual number of bytes used will be significantly reduced.

1) Topology Encoding: Let us consider a random topology. Classical random graphs, also known as Erdös-Rényi graphs, are defined by their degree distribution $P(k) = \exp(-\lambda)\lambda^k/k!$. To uniquely identify N nodes in a ER topology, we need $log_2 N$ bits. To reduce the number of bits, we propose an encoding scheme based on logical labels assigned to neighbors. Let M be the average number of neighbors of a node in a random graph. In our simple encoding scheme, a node assigns a logical label to each of its neighbor, from 0 to M-1. The logical label is independently assigned by each node. When a node joins the network and shares its neighbor information, it includes the logical label corresponding to the neighbor identifier. The source node can now compute the route in terms of the labels. Consider the example graph in Figure 3. Let node A be the sender of a packet to node D. Each node assigns a logical label to its neighbor which is indicated next to the edge connecting the node to the respective neighbor. For example, B assigns the logical labels 0, 1, 2, 3 and 4 to neighbors E, G, C, A and F respectively. Under our encoding scheme, the path from A to B is indicated by *2-2-3*. In a random graph $M \ll N$ [24].

This path encoding scheme saves a considerable amount of space. Consider for example an IP network where *M* = 1000. Then our scheme will represent a savings of over 66%. To put things in perspective, the maximum value of *M* in the current Internet is around 2000 [25].

It is well known that the average path length (APL) of ER networks scale in proportion to ln *N* [26]. In [27] it has been analytically shown that the APL is given by

$$APL = \frac{lnN - \gamma}{ln(pN)} + \frac{1}{2}$$

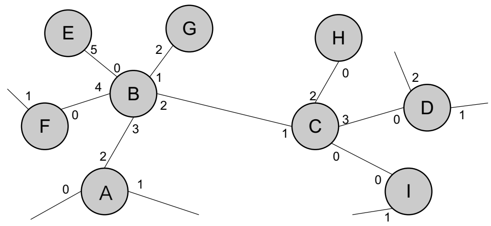
where $\gamma \simeq 0.5772$ is the Euler's constant and $pN = \langle k \rangle$ The APL for ER graphs with $\langle k \rangle$ = 4, 10 and 20 is equal to 6, 4 and 3 respectively for a network with *N* = 10,000. Correspondingly, the diameter *d* of a graph, defined by the maximal distance between any pair of vertices is given by ln *N*/ln(*pN*). From a ER graph of infinite nodes, it has also been shown that 90% of the nodes have an average degree of 5 while less than 0.00001% of the nodes have an average degree of 200 [28].

From the above statistics, we can assume that

- a route computed by the source to any node will be bounded by a few hops in a reasonably large network ($\leq$ 6 for a 10000 node network), and
- the majority of the logical labels in the path will be small numbers.

Our objective is to encode this path with a bit pattern that is quite efficient, is extremely fast to decode and leverages the statistical insights in making the *common case fast* (Amdahl's law). We use the following technique. All labels are classified into two groups Δ_0 and Δ_1. Labels that are less than 32 bits belong to the former group while all others belong to the latter. Accordingly, items in Δ_0 and Δ_1 are encoded by 5 and 8 bits respectively. To distinguish between the labels belonging to the two groups in the path, we prefix a signature of 10 bits in which a 0/1 indicates if the current label belongs to Δ_0/Δ_1. The choice of a fixed length encoding and a signature bit pattern was to facilitate the use of a simple hardware circuit that will allow extremely fast packet forwarding. In order to obtain empirical proof of our approach, we generated different ER networks with up to 10000 nodes. For each topology, we set the average degree to be 4, 10 and 20. We then computed the shortest path between all pairs of nodes and used our encoding technique to compute the number of bits. As can seen in Figure 4, the increase in the number of bits is definitely *O*(ln *N*).

Although extremely unlikely, it is still possible that a label has a value of more than 255. In this case, we use a hop-by-hop mechanism to negotiate part of the path for which the labels are large. Let a path from node n_1 to n_k be denoted by n_1,n_2, ..., n_i, *X*, $n_(i+1)$, ... n_j, *Y*, *Z*, $n_(j+1)$, ..., n_k, where *X*, *Y* and *Z* are labels grater than 255. In this case, the n_1 computes the partial route to the intermediate node n_i. At n_i, the packet falls back to the default hop-by-hop forwarding mechanism to reach *X*. *X* now computes the partial path to n_j. From here, again the packet gets forwarded one hop at a time up to *Z*. Finally, *Z* computes the rest of the path to n_k. Clearly, falling back to the hop-by-hop forwarding mechanism



Node	IP Address
A	A1.A2.A3.A4
B	B1.B2.B3.B4
C	C1.C2.C3.C4
D	D1.D2.D3.D4
E	E1.E2.E3.E4
. . .	. . .

	Route from A to D
Unencoded	A1.A2.A3.A4, B1.B2.B3.B4, C1.C2.C3.C4, D1.D2.D3.D4
Encoded	3, 2, 3

Fig. 3. Path encoding: example graph

compromises the speed, However, this is a tradeoff we chose to allow scalability of source routing. Considering that the likelihood of this happening is 1 hop for 10000 nodes, we believe that this will not adversely affect the performance of the system.

Algorithm 1 Encoder

```
bit_vector final_path[]
bit_vector signature[10] = "0000000000"
array path_in_label[] = find_route(src,dest)
for each position i in path_in_label[] do
  if path[i] > 255 then
    clear remaining path
    break;
  end if
  if path[i] < 32 then
    decimal_to_binary_5_bits(path[i])
    concatenate_with_final_path
    signature[i] = 0
  else
    decimal_to_binary_8_bits(path[i])
    concatenate_with_final_path,bin_path)
    signature[i] = 1
  end if
end for
concatenate(signature,final_path)
```

From the above, we can see that the maximum number of bytes to encode the route in a 10000 node network is about 5. A VSN is essentially a hierarchical resource-rich sensor network. It is hard to imagine it to scale geographically (like the Internet) or in node density (like wireless mote-like sensornets). The space overhead for a 10000 node network is about 50 KB which is modest by current technology standards. Let us consider a TCP/IP packet with a MTU of 1500 bytes. Let the TCP and IP headers each be 20 bytes. For simplicity, let us consider the rest of the packet to contain data. Then an overhead of 5B represents a reduction of less than 1% in the goodput of the system. This shows that source routing is

Algorithm 2 Decoder

```
bit_vector[] recvd_signature = split(path, 1^st 10 bits)
bit_vector[] remaining_path = split(path, remaining bits)
if remaining_path.empty() then
  if I am destination then
    return
  else
    bit_vector final_path[]
    bit_vector signature[10] = "0000000000"
    current = my_ip;
    i = 0;
    while true do
      next_hop = get_next_hop(current, dest)
      if next_hop > 255 then
        if current == my_ip then
          send_pkt(next_hop);
          return;
        else
          break from while loop
        end if
      else
        encode(rest_of_the_path);
      end if
      current = next_hop;
      increment i
      if current = dest then
        break from while loop
      end if
    end while
    concatenate(signature,final_path)
  end if
else
  if recvd_signature[pkt_hop_count] == 0 then
    first_hop = binary_to_decimal_5_bit(remaining_path);
    delete_first_5_bits(remaining_path);
  else
    first_hop = binary_to_decimal_8_bits(remaining_path);
    delete_first_8_bits(remaining_path);
  end if
end if
send_pkt(first_hop);
```

definitely practical in relatively large sized VSN networks.

2) Engineered topology: The above section demonstrates the feasibility of using source routing in ER graphs. However, a VSN will be very likely be a well-engineered system. In this section, we argue that if a VSN is built to exhibit the properties of a scale-free network, then it makes using source routing more practical by making the APL almost constant.

In mathematics and physics, a small-world network is a type of mathematical graph in which most nodes are not neighbors of one another, but most nodes can be reached from every other by a small number of hops or steps. Many empirical graphs are well modeled by small-world networks. Social networks, the connectivity of the Internet, and gene networks all exhibit small-world network characteristics. Small-world

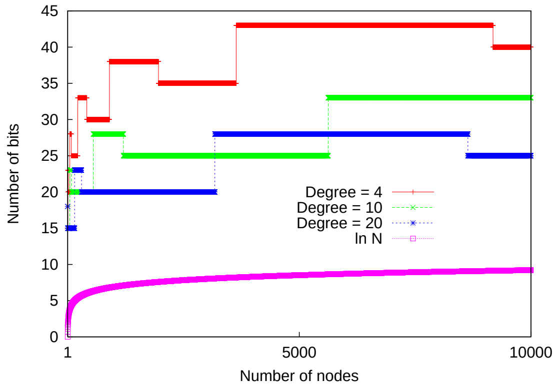


Fig. 4. Scalability of encoding

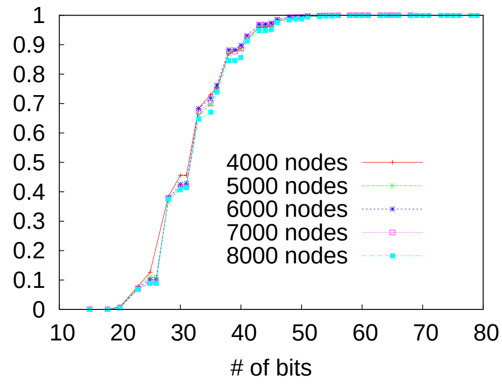


Fig. 5. CDF: WITH encoding

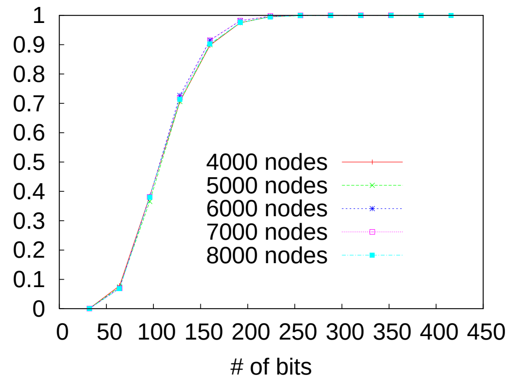


Fig. 6. CDF: WITHOUT encoding

networks have high representation of cliques, and subgraphs that are a few edges shy of being cliques. The highest-degree nodes are often called "hubs". If a network has a degree-

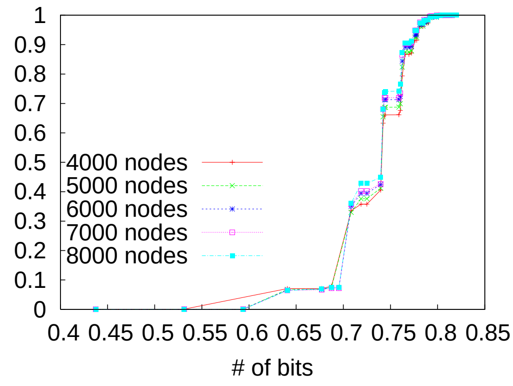


Fig. 7. CDF: Ineffciency

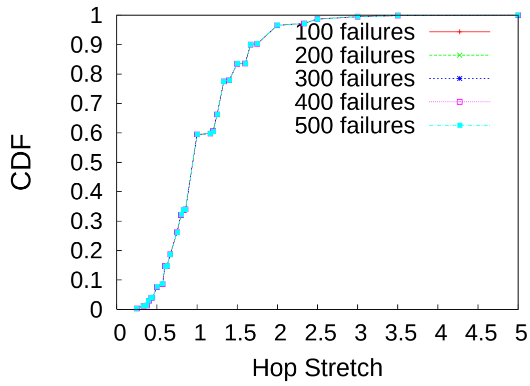


Fig. 8. Effect of number of failures

distribution which can be fit with a power law distribution, it is taken as a sign that the network is small-world. These networks are known as scale-free networks. The probability *P(k)* that a node in the network connects with *k* other nodes is proportional to $k^{-\gamma}$. The coefficient γ may vary approximately from 2 to 3 for most real networks [29]. It was proved that an uncorrelated power-law graph having $2 < \gamma < 3$ will also have a network diameter *d* that is proportional to *O*(lnln*N*) [30].

So from the practical point of view, the diameter of a growing scale-free network might be considered almost constant while the diameter of a 1 million node network is approximately 3 hops.

The power law distribution of a scale-free network produces a hierarchical topology with a fault tolerant behavior. Since failures occur at random and the vast majority of nodes are those with small degree, the likelihood that a hub would be affected is almost negligible. Even if such event occurs, the network will not lose its connectedness, which is guaranteed by the remaining hubs. This characteristic was exploited in generating highly tolerant sensor node placement [31]. As has been mentioned before, the network diameter of growing scale-free networks is almost constant and is usually in the order of 3 to 5 hops for networks with millions of nodes. The bounded hop count significantly reduces packet transmission time thereby improving system latency. In addition, it also solves the scalability problem of source routing.

B. Network dynamics - Lazy Correction

A second problem of source routing is that it is oblivious to network dynamics. In a dynamic routing protocol, when the forwarding decision is made at each hop that has been updated with link state information, the routing protocol *proactively* avoids traversing a link that has failed. A packet that is source routed obviously cannot take advantage of this fact. Thus, a packet might end up in a "dead end" where the next hop mentioned in the source route is no longer reachable.

To remedy this situation, we propose a lazy correction scheme. In this strategy, a packet is allowed to move on along the source specified route until it reaches a node n_i which can not reach the next hop $n_{(i+1)}$. In this case, n_i simply discards the rest of the route and recomputes the remainder of the path from itself to the destination. The packet is forwarded along the new route. This is visually similar to geographic routing around holes in wireless sensor networks. Since n_i is closer to the destination than the source, it will have a more recent update on the state of the links between itself and the destination as compared to the source. If n_i is unable to find a path, the packet is dropped.

Our approach does not affect the convergence and loop-free characteristics of the underlying routing protocol. Our strategy affects where the rerouting decision will be made and not the route itself. However, it definitely could affect the path stretch. Let p_{source} and $p_{dynamic}$ be the paths using source and dynamic routing respectively. We define the path stretch $\lambda_i$of a packet *i* as:

$$\lambda_i = \frac{p_{source}}{p_{dynamic}}$$

The average path stretch $\Lambda$is the average over all packets.

$$\Lambda = \frac{1}{m}\sum_{i=1}^{m}\lambda_i$$

To check this we ran two sets of experiments on our G_0 topology. We induced up to 500 link failures using model Γ_0 and routed 1000 packets between all nodes that had at least one failed path on their source route. Figures 8,9 shows the cumulative distribution frequency. In the second set of experiments, we kept the number of failures fixed at 500 while 1000, 10000 and 50000 packets were similarly routed between all nodes that had at least one failed link. As can be seen, about 90% of the paths had a hop stretch of 1.5 or less. Tables 3 and 4 show the average increase in path length. As exepected, we see that it is likely that a packet will go through a few extra hops. But as will be shown in Section 8, the minimization of the processing and queueing delays at each node, more than offsets this extra cost.

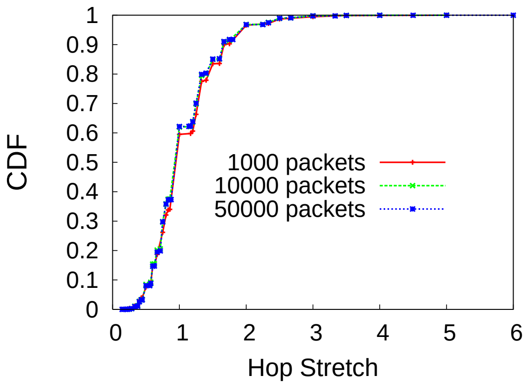


Fig. 9. Effect of number of packets

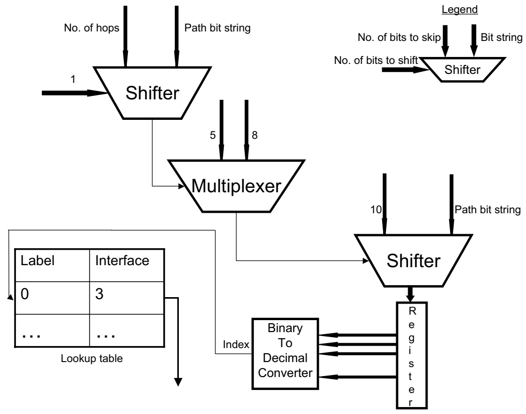


Fig. 10. Short circuiting hardware

TABLE III
AVERAGE PATH STRETCH - 4000 NODES, 1000 PACKETS BETWEEN ALL PAIRS OF NODES

Failures	Average path stretch
100	1.13381
200	1.13414
300	1.1338
400	1.129
500	1.13

VI. SHORT CIRCUITING

Using source routing provides an opportunity to reduce processing delays at intermediate nodes. However, the benefit can only be had if the packet can be forced to bypass the network layer by being "short circuited" through a lower layer. The idea is to provide an alternate datapath for packets in the node to leverage source routing.

Recall that the source path is specified in terms of labels which is then encoded using fixed length encoding. Every node maintains a label lookup table which stores the association between a label (and thereby a neighbor) and a physical interface. When a node receives a packet, it does the following.

- Shift the next bit from the signature string.
- If the string is empty, forward the packet to higher layers.
- If the bit is 0, shift out the next 5 bits from the path string. Decode it to generate the label number.
- Otherwise, shift out the next 8 bits from the path string. Decode it to generate the label number.
- Use the label number as an index to the label lookup table. Enqueue the packet in the queue corresponding to the chosen physical interface.

A simple hardware circuit using standard components is shown in Figure 10.

TABLE IV
AVERAGE PATH STRETCH - 4000 NODES, 500 FAILED LINKS

Number of packets	Average path stretch
1000	1.13
25000	1.0981
50000	1.09884

VII. APPROXIMATE FAIR QUEUEING

An important component that affects the packet latency is the packet scheduling algorithm used by routers in the network. The packet scheduler determines the order in which packets of various independent flows are forwarded on a shared output link. Packet scheduling affects the queueing delay. A poorly designed scheduling algorithm could add as much as two seconds of delay to a packet [32] In general, a packet scheduler should have the following properties.

Fairness: The packet scheduler must provide some measure by which multiple flows receive a fair share of system resources, for example, the shared output link. In particular, each flow should get its fair share of the available bandwidth, and this share should not be affected by the presence and (mis)behavior of other flows.

Bounded delay: Multimedia applications require the total delay experienced by a packet in the network to be bounded on an end-to-end basis. The packet scheduler decides the order in which packets are sent on the output link, and therefore determines the queuing delay experienced by a packet at each intermediate router in the network.

Low complexity: While often overlooked, it is critical that all packet processing tasks performed by routers, including output scheduling, be able to operate in nanosecond time frames. The time complexity of choosing the next packet to schedule should be small, and in particular, it is desirable that this complexity be a small constant, independent of the number of flows N.

Designing a packet scheduler with all of these constraints is a big challenge that warrants attention. Generally speaking, packet scheduling algorithms can be divided into two categories. Timestamp schedulers, where packets are transmitted in increasing order of their timestamps, have good delay properties, they suffer from a sorting bottleneck that results in a time complexity of $O(\log F)$ per packet. Round-robin schedulers assign time slots to flows in some sort of round-robin fashion and achieve a complexity of $O(1)$. They tend to have poor delay bounds and output burstiness.

We present a queueing strategy that aims to bridge the gap between the two. Specifically, our algorithm has low complexity, good fairness and predictable single packet latency bound that is independent of the number of flows.

A. Uniqueness of Embedded Networks

A GPS-based scheduling algorithm, for example Weighted Fair queueing (and its many variants) is a good choice for wide area public networks, for example the Internet. However, using the same strategy in an VSN might not work for many reasons.

At the heart of WFQ algorithm is the assignment of weights to flows. This is a very difficult problem and all the results thus far are based on the specification of traffic characteristics by the sender [33]. Characterizing traffic in sensornets can be a bit impractical as these networks react to unpredictable phenomena in the physical world, for example terrorist attacks, which cannot be accurately modeled.

The GPS-like algorithms use the notion of a *flow* to make scheduling decisions. The definition of flow is very fuzzy. It is commonly used to refer to a series of packets from a source to a destination. But a source/destination can be defined in terms of its network address, the particular application in question and the port. Correspondingly, a flow can be define in terms of one or many combinations of the aforesaid. The concept gets murkier in sensornets because many different kinds of processing can take place inside the network. To allow better utilization of resources, the system might decide to opportunistically *merge*, *process* or *split* data streams [34]. Without a strict definition of a flow, it is hard to understand how a GPS algorithm might be implemented.

Finally, a wide area network typically have two broad classes of traffic. The *elastic* class refers to those that can tolerate delays, for example file transfers. The *priority* traffic refers to delay sensitive applications that have soft real time constraints, for example multimedia data. In a VSN, all traffic belongs to the latter class. Thus, there is no scope of giving preference to any particular class making it hard to implement class-based fairness.

B. Approximate Fair Queueing

We now present Approximate Fair Queueing (AFQ), a simple scheduling algorithm that addresses the above mentioned issues. The GPS scheduling algorithm is a generalization of the *bit-by-bit round robin* (BR) algorithm. In the BR scheme, each flow sends one bit at a time in round robin fashion. In Approximate Fair Queueing, packets are classified according to their weights into different queues and each queue is scheduled in a round robin fashion. Specifically, each router maintains M queues where M is a constant. Let the maximum transmission unit (MTU) of the underlying physical layer be B bytes. A queue Q_i is assigned an **integral** weight ϕ_i, where $0 < \phi_i \leq B$ An incoming packet p is assigned to Q_j if the size of p is greater than ϕ_{j-1} but less than or equal to ϕ_{j+1}. Scheduling proceeds in a round robin manner among the M queues. Let ϕ_{max} be the maximum weight assigned to any queue. Then while servicing Q_j, the scheduler will send a burst of $\lfloor \frac{\phi_j}{\phi_{max}} \rfloor$ packets.

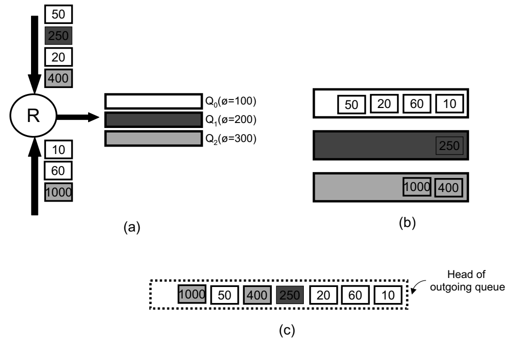


Fig. 11. (a) R is a router with 2 input links and 3 queues with weights = 100,200,300 (b) Packets are assigned to the different queues based on their sizes. (c) The exact schedule of packets assuming that the round starts with Q_0

For example, let a router R maintain 3 flows where ϕ_0 = 50, ϕ_1 = 500 and ϕ_2 = 1000. A packet of size 50 bytes will be placed in Q_0, one of 250 bytes will be placed in Q_1 while one of 1000 bytes will be placed in Q_2. The scheduler will then schedule 30, 3 and 1 packets from queues Q_0, Q_1 and Q_2 respectively. If a queue does not have any packet, the scheduler moves on to the next one. An example with a schedule is shown in Figure 11.

The classification of weights can be done at a very fine scale. A fine grain classification maintains B queues with $\phi_0 = 1$, $\phi_1 = 2$ and so on. This provides the most fairness according to our scheme and we refer to this as the ideal scenario. However, as has been shown in many studies, multimedia traffic typically peaks around 40 bytes or 500 bytes or 1500 bytes [37]. Using that insight, we instead suggest a much coarser classification. The system maintains 5 queues: $\phi_0 = 100$, $\phi_1 = 500$. $\phi_2 = 600$, $\phi_3 = 1000$ and $\phi_4 = 1500$. This is for an Ethernet based system which typically has an MTU of 1500 bytes. Notice that the coarse-grained approach approximates the ideal scheme, hence the word "Approximate" in the name AFQ.

AFQ is amenable to a very simple hardware implementation. A modulo-$\log_2 M$ binary counter is required for a round robin selection of the queues. For each queue, another modulo-$\log_2(\lfloor \frac{\phi_j}{\phi_{max}} \rfloor)$ counter is needed to keep a track of the number of packets sent in the current round. This implementation satisfies two of our requirement - the scheduling algorithm should be fast and be easy to implement.

In order to verify the fairness of our algorithm, we routed up to F (= 100) flows through a single node with one output link. The flow characteristics were randomly chosen from Table 2. We compare Wf2q (a variant of WFQ with better worst case bounds), a fine-grained AFQ, a coarse-grained AFQ and a *two-level* AFQ (2L-AFQ). 2L-AFQ is a variation of AFQ in the presence of a notion of flows. The router now maintains two

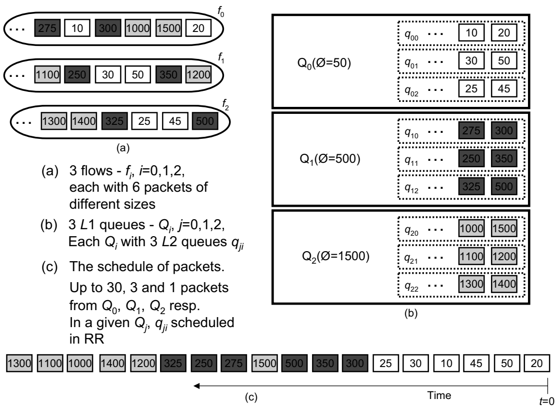


Fig. 12. Two level AFQ scheduling

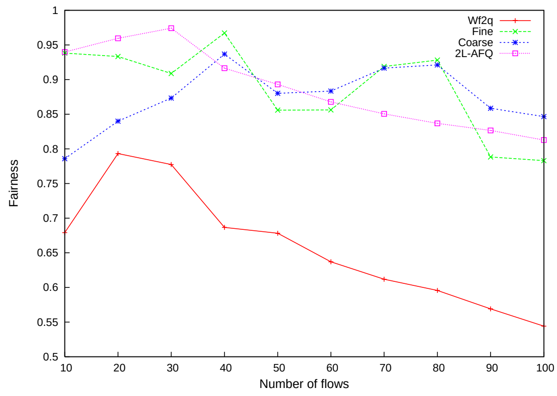


Fig. 13. Comparison of fairness

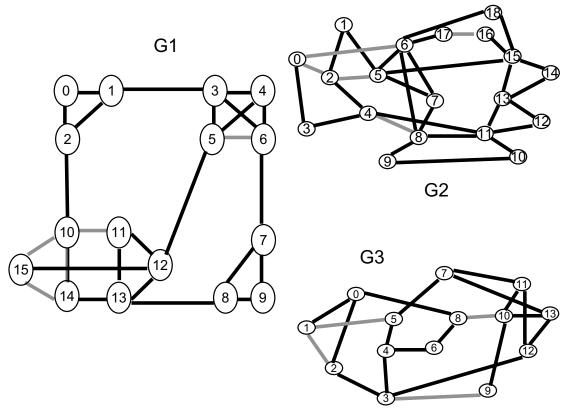


Fig. 14. Topologies used in evaluation

sets of queues. The first level ($L1$) queue is based on packet size. In the second level, we maintain additionally a queue ($L2$) for each flow. The scheduler first chooses an ($L1$) queue that has packets to send. Based on its weight, let n be the number of packets that can be sent. These n packets are now chosen in a round robin manner from the ($L2$) queues that are inside the chosen ($L1$) queue. Figure 12 demonstrates this algorithm. As is shown in the diagram, the node maintains three $L1$ queues and each $l1$ queue in turn maintains three $L2$ queues. When Q_1 is chosen, the scheduler selects 3 packets to send. It selects one packet each from $L2$ queues q_{10}, q_{11} and q_{12} respectively.

The metric that we use for comparison is widely known Jain's fairness [35]. If t_i is the throughput of flow F_i, then Jain's fairness is defined as:

$$Fairness = \frac{(\sum t_i)^2}{(F \times \sum x_i^2)}$$

with 1 being the most fair and 0 being the least. As can be seen from Figure 13, all variations of AFQ are fairer than Wf2q. In addition, the latter suffers a greater reduction in its fairness index as the number of flows increase.

VIII. Evaluation

In this section, we put together all the concepts discussed in the previous sections and compare it with a traditional packet forwarding system. We implement three packet delivery systems. The first one uses source routing with lazy correction, a link level hardware forwarding and 2L-AFQ scheduling strategy. We refer to this as "Two level". The second is essentially the same as the first with the difference that it uses a coarse grained AFQ strategy instead. We refer to this as "Coarse". The third simulates a regular packet switched network using dynamic routing and the worst-case weighted fair queueing scheduling algorithm [4] We refer to this as the the Wf2q.

A. Simulation parameters

We use the topologies G_1, G_2 and G_3 as shown in Figure 14 with their properties being discussed in Table 1. The grayed out links are the ones that fail according the failure model Γ_1. Each topology has a single destination node. All other nodes send data packets to this chosen node. The chosen destination node for G_1, G_2 and G_3 are 15, 0 and 1 respectively. Each source node generats one flow. A flow is randomly chosen from one of the four types shown in Table 2. Each flow generates 20,000 packets. The size of a packet is randomly chosen from the intervals [40B,200B], [400B,600B] and [1000B,1500B]. Packets are generated at a fixed rate calculated from the type of flow.

The delay of a packet is modeled as follows. When a packet reaches a node, it can either go through the lower layers and then be handled by the IP layer, or it can be short circuited through the link layer using the hardware circuit described in Section 6. In both cases, the fundamental task is a table lookup to select the next hop or the physical interface respectively. IP lookup can be implemented in many different ways. To make a fair comparison with our short circuit switch, we assume that it is implemented in hardware. So for both strategies, we assign a constant time of 100 ns for table lookup. The penalty of the former comes from the traversal of the network layers.

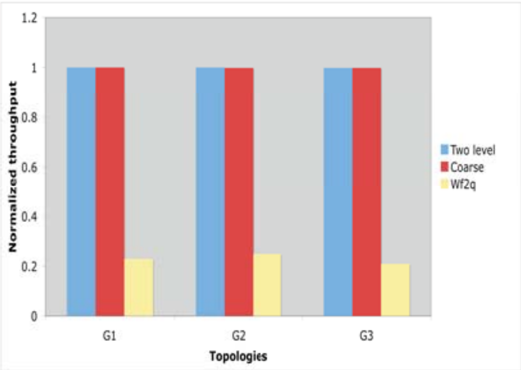


Fig. 15. Average throughput

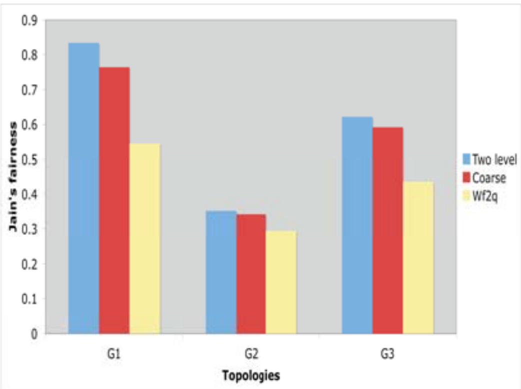


Fig. 16. Jain's fairness

For this, we assign a fixed value of 1 ms. The packet then waits in a queue, the duration of which is decided by the dynamic traffic conditions. To each packet, we then add a delay proportional to the efficiency of the queueing algorithm. Recall that a Weighted Fair Queueing algorithm has a running time complexity of $O(F)$, where F is the number of flows. To simulate realistic scenarios, we inject 10000 dummy flows at each node. Based on that, we assign a delay of 3ms to the execution of the Wf2q algorithm. By contrast, AFQ has a constant time complexity in $O(1)$. We assign a delay of 1ms in executing the AFQ algorithm. Finally, the transmission and propagation delays are set as described in Section 3.

B. Metrics

We evaluate our approaches based on the following metrics.

- **Throughput:** We provide a raw measure of the efficiency of the system in terms of its throughput. We provide two measures of throughput. The *per-flow* throughput is the number of bits successfully transmitted per second The *average* throughput takes the weighted harmonic mean of the average per-flow throughput [35], using the message length as the weight:

$$average\ \ throughput = \frac{\sum_{i \in F} b_i}{\sum_{i \in F} t_i}$$

where F is the total number of flows in the network, b_i represents the number of bytes sent over connection i and t_i its duration.
- **Latency:** We provide a holistic measure of application latency that we refer to as the *playback* latency. A multimedia stream has a strict deadline within which successive packets must be received. Packets arriving after the deadline are considered useless and are treated as lost; and the loss of a certain number of packets will seriously affect the quality of the application. Playback latency is a measure of how long the destination has to wait before it can start playing the stream without losing a single packet due to jitter violations. More specifically, for every flow, let p packets be generated at a the rate of r packets/second starting at time t_s. Let t_f be the time that the final packet is received. Then,

$$playback\ latency = t_f - (p * r) - t_s$$

In addition, we also present the cumulative distribution frequencies of all packets of all flows.
- **Fairness:** We compute Jain's fairness index as described in Section 7.2 using the average throughput as of each flow as described above.

C. Results

Figure 15 shows the average throughput of the three systems normalized with respect to the coarse-grained approach. Across all topologies, our system have a much better throughput than Wf2q system. At the same time, the fairness index is also better, as shown in Figure 16. A more interesting insight can be had when comparing the throughputs of the individual flows. Even the per-flow throughput is better in our system. The variation of the throughputs is lower in case of Wf2q suggesting that it might provide tighter, but not necessarily better, on the throughput variance. The trend, however, is very similar in all systems across all topologies. The playback latency data, on the other hand, is quite surprising. In G_1, the performance of Wf2q varies by almost 10 times. Our approach, on the other hand, remains practically constant. The same trend for Wf2q is observed in G_3 where our approach has a variation of a factor of 4. In G_3, the expected behavior changes for certain flows where Wf2q performs better. If we look at the CDF of latencies of all packets across all flows in G_1, our system shows that more than 90% of packets have a latency of 200s. The corresponding number is 1800s for Wf2q. In G_2, Wf2q performs sightly better. 90% of packets have a latency of 130s while in our system the corresponding number is 150s. Finally, in the G_3, topology, the cumulative latency in our system is almost half that of the Wf2q system. In all experiments, we observe that there is not much difference between the 2L-AFQ and the coarse-grained AFQ approaches. This suggests that an approximate approach is comparable to

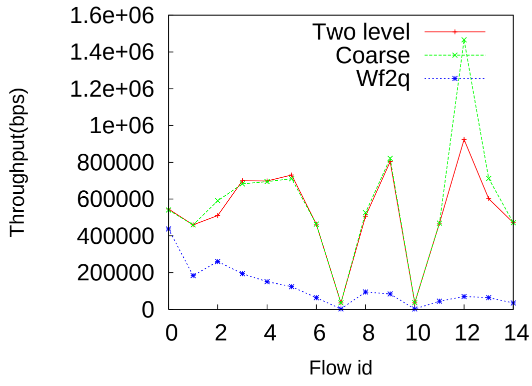


Fig. 17. Per-flow throughput: G1

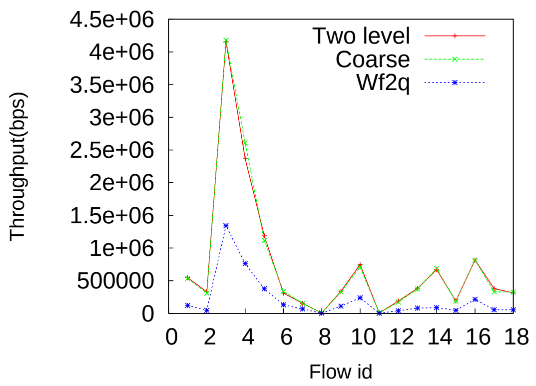


Fig. 18. Per-flow throughput: G2

the more accurate algorithm. This perfectly suits a VSN where the nodes get sudden bursts of packets with the route and the packet size as the only information to exploit.

From the above experiments we draw the following conclusions.

- Source routing with AFQ and link level short-circuiting provides improves packet latencies over IP-based networks using dynamic routing. This also improves the throughput of the system.
- It is fair to all flows as the number of flows in the system increases.
- It's behavior is fairly agnostic to network topologies.

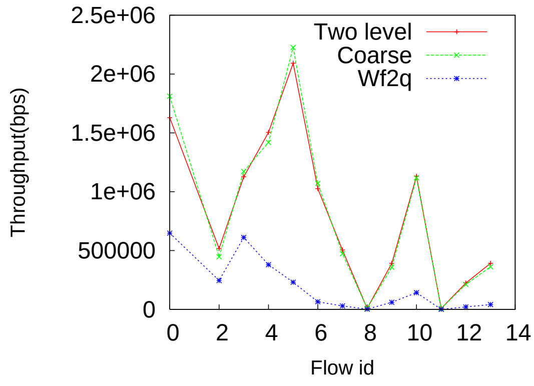


Fig. 19. Per-flow throughput: G3

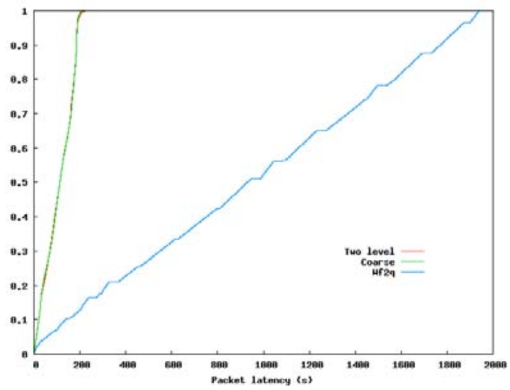


Fig. 20. CDF - Packet latencies: G1

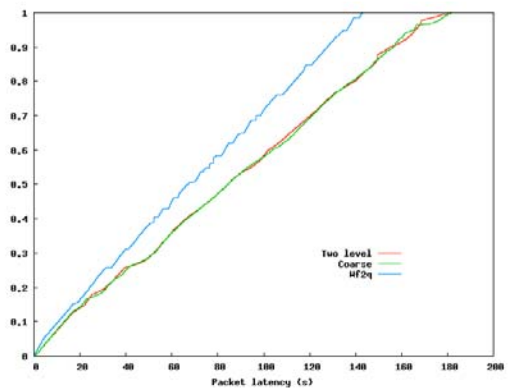


Fig. 21. CDF - Packet latencies: G2

IX. Conclusion

In this paper, we have shown that inspite of its perceived shortcomings, source routing is actually usable in practical sized networks. If VSNs are built as scale-free networks and if a topology is encoded in terms of labels, then source routing imposes only a modest overhead on storage requirements and goodput performance. Source routing allows routing layer functionality to be bypassed at intermediate nodes. We presented a hardware architecture that will allow short-circuiting of a packet through the link layer. We further reduce the packet latency at nodes by employing a fair but extremely fast packet scheduling algorithm. All the above strategies working in concert provide a significant improvement of packet latency

World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:3, No:4, 2009

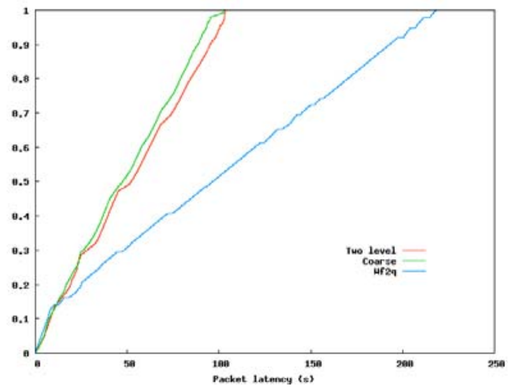


Fig. 22. CDF - Packet latencies: G3

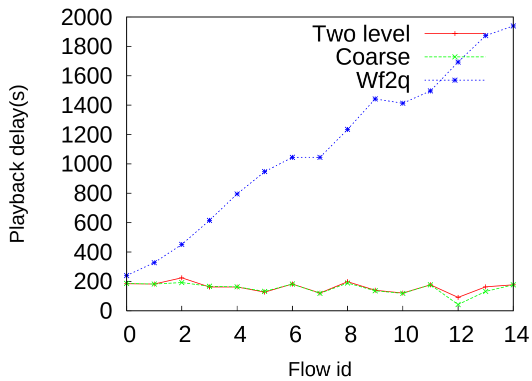


Fig. 23. Playback latency: G1

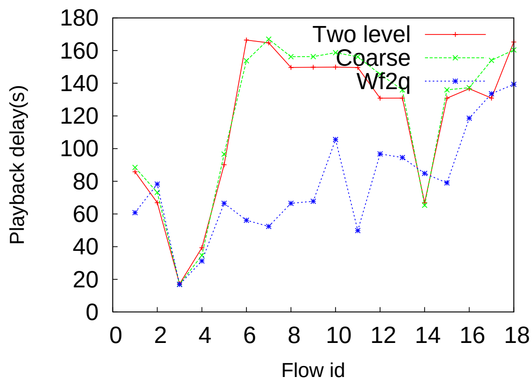


Fig. 24. Playback latency: G2

over a similar IP-based network.

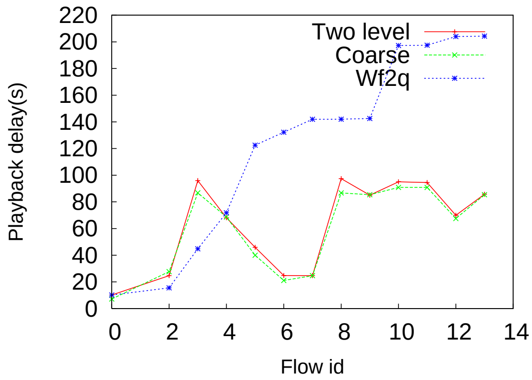


Fig. 25. Playback latency: G3

REFERENCES

[1] "http://www.airport-int.com/categories/surveillance-solutions/peoplemover-project-brings-21st-century-surveillance-system-to-dallas-airport.asp."

[2] "http://www.tropos.com/pdf/case_studies/tropos_casestudy_new_orleans.pdf."

[3] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queuing algorithm," in *SIGCOMM*, 1989.

[4] J. B. H. and Zhang, "Wf2q : Worst case fair weighted fair queuing," in *INFOCOM*, 1996.

[5] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round robin," in *SIGCOMM*, 1995.

[6] N. Figueira and J. Pasquale, "Leave-in-time: A new service discipline for real-time communications in a packet-switching network," in *SIGCOMM*, 1995.

[7] S. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *INFOCOM*, 1994.

[8] G. Chuanxiong, "An o(1) time complexity packet scheduler for flows in multi-service packet networks," in *SIGCOMM*, 2001.

[9] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture: an overview," *Internet RFC 1633*, June 1994.

[10] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," *Internet RFC 2475*, December 1998.

[11] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," *Internet draft ¡draft-ietf-mpls-arch-01.txt¿*, March 1998.

[12] X. Xipeng and L. Ni, "Internet qos: a big picture," *IEEE Network*, March 1999.

[13] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case," *IEEE/ACM Transactions on Networking*, vol. 1, 1993.

[14] J. Xu and R. Lipton, "On fundamental tradeoffs between delay bounds and computational complexity in packet scheduling algorithms," in *SIGCOMM*, 2002.

[15] L. Lenzini, E. Mingozzi, and G. Stea, "Aliquem: A novel drr implementation to achieve better latency and fairness at o(1) complexity," in *IWQoS*, 2002.

[16] S. .Cheung and C. Pencea, "Bsfq: Bin sort fair queuing," in *INFOCOM*, 2002.

[17] S. Ramabhadran and J. Pasquale, "The stratified round robin scheduler:design, analysis and implementation," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. 6, December 2006.

[18] T. Fuhrmann, "The use of scalable source routing for sensor networks," in *2nd IEEE workshop on Embedded Sensor Networks*, 2005.

[19] V. Hadimani and R. Hansdah, "An efficient distributed scheme for source routing protocol in communication networks," *Lecture notes in Computer Science*, vol. 3347, November 2005.

[20] C. Glass and L. Ni, "The turn model for adaptive routing," in *ISCA*, 1992.

[21] "Moving picture experts group, october 2006 http://www.chiariglione.org/mpeg."

[22] "Dolby laborotories inc. www.dolby.com."

[23] A. Markopoulou, G.Iannaccone, S.Bhattacharyya, C.N.Chuah, Y.Ganjali, and C.Diot, "Characterization of failures in an operational ip backbone network," *IEEE Trans. on Networking*, vol. 16, October 2008.

[24] D. J. Watts and S. H. Strogatz, "Collective dynamics of small world networks," *Nature*, vol. 393, no. 6684, pp. 440–442, June 1992.
[25] "http://www.caida.org/research/topology/as_core_network/."
[26] M. et al., *Phys. Rev. E*, vol. 64, no. 026118, 2001.
[27] A. Fronczak, P. Fronczak, and J. A. Holyst, "Average path length in random networks," *Phys. Rev. E*, 2002.
[28] M. Stumpf and C. Wiuf, "Sampling properties of random graphs: The degree distribution," *Phys. Rev. E*, vol. 72, 2005.
[29] H. Seyed-allaei, B. Ginestra, and M. Marsili, "Scale-free networks with an exponent less than two," *Phys. Rev. E*, vol. 73, 2005.
[30] R. Cohen and S. Havlin, "Scale-free networks are ultrasmall," *Phys. Rev. E*, vol. 90, 2003.
[31] M. Ishizuka and M. Aida, "The reliability performance of wireless sensor networks configured by power-law and other forms of stochastic node placement," *IEICE Trans. on Communications*, vol. E87-B, no. 9, pp. 2511–2520, September 2004.
[32] J. Davidson, M. Bhatia, S. Kalidindi, S. Mukherjee, and J. Peters, *VoIP: An In-Depth Analysis*. Cisco Press, 2006.
[33] P. Lieshout, M. Mandjes, and S. Borst, "Gps scheduling:selection of optimal weights and comparison with strict priorities," in *ACM SIGMETRICS Performance Evaluation Review*, 2006.
[34] A. Ghosh and T. Givargis, "A software architecture for accessing data in sensor networks," in *INSS*, 2008.
[35] R. Jain, *The Art of Computer Performance Analysis*. Wiley, 1991.