Link-State Routing Protocol for Multi-Channel Multi-Interface Wireless Networks

Cheolgi Kim*, Young-Bae Ko† and Nitin H. Vaidya*

*Dept. of Electrical & Computer Engineering, Univ. of Illinois at Urbana-Champaign, Urbana, IL
Email: {cheolgi, nhv}@uiuc.edu
†Dept. of Information & Computer Engineering, Ajou University, Suwon, Korea
Email: youngko@ajou.ac.kr

***Abstract*—A lot of military networks maintain multiple wireless channels and exploit frequency-hopping spread spectrum on those channels for anti-jamming. One drawback of using multi-channel communications is the high overhead involved in broadcast operations: a transmitter should transmit a broadcast packet to all channels that are possibly occupied by a receiver. This makes certain broadcast-intensive mechanisms, such as, link-state routing difficult to implement. Link-state routing, however is faster and robust, which makes it suitable for military applications. In this paper, we present a link-state routing protocol tailored for multichannel networks by minimizing the broadcast overheads. This is achieved by means of a special set of nodes called cluster-heads. We have implemented our protocol on a multichannel, multi-interface wireless test bed and have compared its performance with an AODV-like reactive routing protocol, which is also tailored for multi-channel multi-interface networks. The measurements on our test bed show that the proposed link-state routing protocol provides transient communications in a comparable or better performance.**

I. Introduction

In wireless networks, multiple communication links share the same spectrum as communication media. This leads to increased levels of channel contention and interference. Multi-channel approaches help manage such interference and contention among the nodes by slicing the spectrum into multiple channels and assigning them to the participating nodes. They bring down per-channel activity of communication and per-channel bandwidth to manageable level, which assist protocol design and implementation. Moreover, narrow per-channel bandwidth can simplify circuit design, too.

Military networks use the frequency hopping spread spectrum techniques, which naturally provide multiple channels of communications, to mitigate the effect of jamming interference. Multiple channels, however, partition the network based on the channel used. This may result in a disconnected network if the nodes communicate only in their assigned channels. To resolve this problem, several multi-channel ad-hoc / mesh network approaches have been proposed in the literature [1], [2], [3]. Furthermore, several routing mechanisms are proposed for multi-channel networks. In this paper, we discuss a link-state routing mechanism that is suitable for a multi-channel, multi-interface wireless network.

The work in this paper was funded by The Boeing Company.

In link-state routing, each node maintains the link-state information of the entire network and makes route decision based on this information. Even though the overheads due to link-state information exchange are relatively high, route is discovered promptly without additional control overhead such as route request packets. Thus, link-state routing makes communication reliable, making it suitable for high-availability systems, such as military networks.

In this paper, we introduce MCLSR (Multi-Channel Link-State Routing) Protocol, which is designed for multi-channel multi-interface wireless networks for high-availability systems. Most of the existing designs of the link-state routing exploit the broadcast nature of the wireless medium for propagating the link-state information [4]. However, since the broadcast overheads in a multichannel network are much higher than a single-channel environment, the existing implementations of the link-state routing protocols are not desirable to be directly applied for multi-channel networks. Notice that each node has to broadcast the packet in all the channels to deliver a single broadcast packet as a neighboring node could be tuned to any of the channels available in a multichannel network, thereby increasing the overhead involved. To minimize the broadcast overheads, we propose to have a set of cluster-head nodes, which are elected dynamically for exchanging the link-state information over the network. The cluster-head nodes collect the link-state information from their dependent nodes, share it with other cluster-heads, and rebroadcast the shared link-state information to its dependents. We explain the procedures involved in Section IV.

We have implemented our protocol on a real multi-channel, multi-interface wireless test bed with two IEEE 802.11a interfaces per node. We discuss the implementation issues and the experimental results in Section V. Our results show that our link-state routing mechanism provides a stable and reliable performance in high traffic conditions. Then, we conclude our paper in Section VI.

II. Related Work

After the concept of wireless mesh networks emerged, several variants of the original link-state routing protocol tailored for wireless networks have been proposed. For instance, Jacquet et al. proposed the Optimized Link State Routing Protocol (OLSR) [4]. In this protocol, each node selects multipoint relay nodes and link-state broadcast is performed only by these selected set of nodes. This, in turn, reduces the amount of broadcast messages in the network. Furthermore, the link-information of only a subset of links are sent out every time. OSPF-MANET (Open Shortest Path First for MANET) is a link-state routing protocol proposed for mobile ad-hoc networks (MANET) by the IETF OSPF working group [5]. OSPF-MANET extends OSPF, which is widely used in wired networks, for wireless networks. It also allows only subset of nodes for broadcast flooding.

Sivakumar et al. proposed CEDAR (Core-Extraction Distributed Ad Hoc Routing) [6], a link-state routing mechanism designed for real-time applications. In CEDAR, selected core nodes perform unicasts between them to propagate link states as our approach. However, its design does not reflect multi-channel characteristic of wireless networks. Draves et al. proposed WCETT (Weighted Cumulative Expected Transmission Time) as a multi-channel multi-interface routing metric, and their link state routing protocol, LQSR (Link-Quality Source Routing) [1]. LQSR is a source routed link-state protocol derived from DSR [7]. They implemented the routing protocol on test beds and compare the performance between routing metrics.

Kyasanur and Vaidya proposed a novel multi-radio wireless network architecture, called Net-X, and a routing metric [3]. The Net-X architecture is used in this paper. They proposed a routing metric and a reactive routing protocol, and implemented them on test beds [8].

III. Network Model

Our network consists of a set of wireless nodes, each of which equips itself with two wireless interfaces. One of these interfaces remain fixed on a channel and we name this interface as the 'fixed interface.' The other interface is capable of switching across channels when required and hence called the 'switchable interface.' We call the channel in which the fixed interface of a node operates as the 'fixed channel' of that node. Only the fixed interface is used for receiving data. Additionally, the fixed interface can be used for transmitting data, if any, on the fixed channel. Transmissions on the other channels are taken care by the switchable interface. Suppose that node u has a packet to node v and they are on different fixed channels. Then, the switchable interface of u is tuned to the fixed channel of v and transmits the packet on that channel. If they are on the same fixed channel, node u's fixed interface is used for the transmission. We therefore need a channel allocation scheme for assigning channels only to the fixed interface as the channel in which a switchable interface operates is determined based on the fixed channel of the next hop node of a flow. If a node has a broadcasting packet, it sends the packet to all possible channels because every channel probably has a neighbor. For this purpose, the switchable interface performs round-robin channel switching through all channels except the fixed channel.

Let $G = (V, E)$ denote a network, where V is the set of nodes in the network, and E is the set of links between the nodes. A link $(u, v) \in E$ exists between two nodes, $u, v \in V$, if they are within the communication range of each other. Let C denote the set of channels to be assigned to the nodes and $\vec{q}(u, v)$ represent the unidirectional link quality of a link $(u, v) \in E$. The bidirectional link quality $q(u, v)$ is defined by $q(u, v) = g(\vec{q}(u, v), \vec{q}(v, u))$. In our implementations it is simply defined as:

$$q(u, v) = g(\vec{q}(u, v), \vec{q}(v, u)) = \vec{q}(u, v) \cdot \vec{q}(v, u).$$

A subset of nodes in V are destined as the 'cluster-head' nodes and we denote this subset as V_C. We explain more on the cluster-head nodes later. Additionally, we assume that every node in the network has two types of neighbors, namely tight neighbors (N_t) and loose neighbors (N_l) depending on the quality of the link with the neighboring nodes. In general, every node maintains two common link threshold values, namely Th_l and Th_t, such that $\mathsf{Th}_l < \mathsf{Th}_t$. N_l and N_t are defined by Th_l and Th_t such that

$$N_l(v \in V) = \{u \mid u \in V \text{ and } q(u, v) > \mathsf{Th}_l\}$$
$$N_t(v \in V) = \{u \mid u \in V \text{ and } q(u, v) > \mathsf{Th}_t\}.$$

Note that $N_t \subset N_l$ because $\mathsf{Th}_l < \mathsf{Th}_t$ should be satisfied. Due to changing link conditions, $N_l(v)$ and $N_t(v)$ change over time. These sets are used for cluster-head selection (see Section IV).

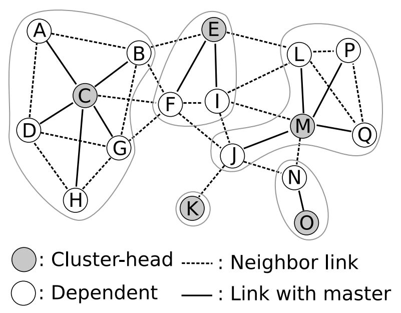
IV. Multi-Channel Link-State Routing Protocol

Multi-Channel Link-State Routing (MCLSR) protocol is a link-state routing protocol for multi-channel multi-interface wireless networks with mission-critical applications. It is tailored to minimize broadcast overhead due to link-state propagation. To meet the requirements of mission-critical applications, all nodes keep the link state information of a network. In typical implementations of wireless link-state routing mechanisms, such as OLSR and OSPF MANET, this is achieved using a broadcast flooding. In our network model, the overhead of broadcast is $|C|$ times larger than that in single-channel networks. Minimizing the broadcast overhead, therefore, is considered one of the primary goals of our protocol.

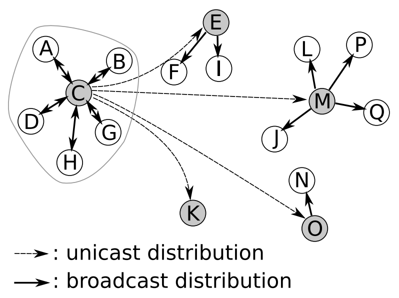
In MCLSR, nodes are classified into two disjoint categories, namely the cluster-heads and dependents. Once a node is chosen as a cluster-head, some of the other nodes that are within one hop from this node become its dependents. Each dependent can have multiple neighboring cluster-heads but one and only one of them is identified as its *master* cluster-head. A cluster-head cannot have another cluster-head as its tight neighbor (N_t) as a rule[1]. A cluster-head along with its

[1]If a pair of cluster-heads indicate that they are tight neighbors of each other, either of them will lose its cluster-head-ship according to the protocol

dependents forms a cluster. A cluster-head is responsible for gathering the link-state information of its cluster based on '*hello messages*,' which are periodically broadcasted by the dependents in the cluster. Then, the cluster-head exchanges the link-states of its cluster members with all the other cluster-heads. The cluster-head receiving link-states of another cluster, broadcasts the link-states. Consequently, link-state of every node is distributed to all the other nodes.



(a) Example topology



(b) Link-state flow from cluster managed by Node C to the network

Fig. 1. Example topology of MCLSR

Let us see an example for details. Figure 1(a) shows an example of a network using MCLSR. Shaded nodes are cluster-heads and the non-shaded nodes are the dependent nodes. A cluster-head and its dependents are connected by solid lines and the other neighboring pair of nodes are connected by dotted lines. Every dependent node is in only one cluster. The cluster has only one cluster-head node, which is a *master* cluster-head of the cluster. All other neighboring cluster-head nodes, if any, are considered as a generic neighbor to a dependent node. Sometimes, a cluster may contain only one node (which will be a cluster-head), as node K in the figure.

Figure 1(b) shows how the link-states are propagated from a cluster node to the entire network. Link states of the cluster members are delivered to cluster-head C through '*hello messages*.' Cluster-head C constructs a '*cluster link-state*' that contains all link-states of the cluster members. Link-state of an individual node is called '*node link-state*' to be distinguished from *cluster link-state*. Each cluster-head periodically distributes the cluster link-states to other cluster-heads through *inter–cluster-head unicasts*, which is presented by dashed arrows in Figure 1(b). The delivered *cluster link-state* is broadcasted to its own cluster members.

Hello messages and *inter-cluster-head messages* are the only control messages in MCLSR. The message formats of the hello messages and the inter–cluster-head messages are discussed in the next section.

A. Node link-state and cluster link-state

Node link-state and *cluster link-state* are the internal information units of delivered information by control messages, such as *hello messages* and *inter–cluster-head unicast messages*. Fig. 2 depicts their internal structure.

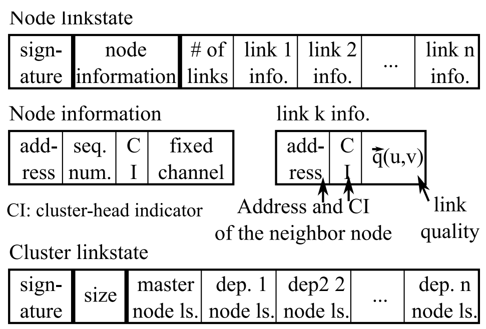


Fig. 2. Link-state information structure

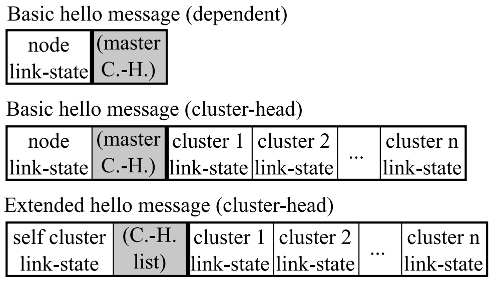
Node link-state: The node link-state is the basic data structure that are contained in both the hello message and the inter–cluster-head message. A node link-state consists of node information that contains the state of the node, and a set of link information that contains the link qualities. Node information consists of a sequence number, which is incremented for every hello message, a cluster-head indicator (CI) bit, which indicates if this node is a cluster-head or not, and the fixed channel used by the fixed interface.

The link information, on the other hand, contains the directional link quality, which is measured from incoming hello messages. Notice that total link quality, which is used by routing protocol as a metric, is a function of a pair of directional link quality, outgoing and incoming. Because each node can only measure link quality of incoming direction, the link quality field of node link-state contains that directional link quality, which is $\vec{q}(u, v)$ where u is the outgoing node, and v is the incoming node. The other directional link quality is collected from node link-state of the node at the opposite side of the link. Then, total link quality is calculated from the pair of directional link qualities. In our protocol implementation, directional link quality is the proportion of successful hello message delivery. And, total link quality of a link is given by $(\vec{q}(u, v) \cdot \vec{q}(v, u))$.

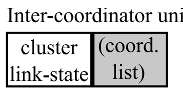
Additionally, the link information contains a cluster-head indicator (CI) bit, which is used to indicate whether the node on the other side of the link is a cluster-head or not. The CI bit in the link information helps in cluster-head discoveries.

Cluster link-state: Cluster link-state message consists of the link-states of all the cluster members. It is formed by concatenating the link-state structure of each of the cluster members as shown in Fig. 2.

B. Control messages



(a) Various hello message formats



(b) Inter–cluster-head unicast format

Fig. 3. Control message structures

Hello message: The purpose of the hello message in MCLSR is in three folds:

1) Distribution of up-to-date node link-states
2) Periodic link quality measurement on the receiver side, and
3) Distribution of cluster link-states from a master cluster-head to its dependents.

A dependent node periodically transmits a hello message containing its node link-state. Every dependent node, upon receiving a hello message sent by other nodes in the neighborhood, updates its link-state database up-to-date. Additionally, it measures the directional quality of the link, $\vec{q}(u, v)$ where v is receiving node, from each of its neighbors, u, based on the received hello messages. For measurement accuracy, the size of hello message is fixed to 1024 bytes. Hello messages are zero padded, if required, to satisfy this fixed size criteria. Fig. 3(a) describes the structure of a hello message sent by a dependent node. As seen in the figure, a dependent node prepares a hello message by attaching the address of its master cluster-head to the node link-state information. The acknowledged master cluster-head includes the received node link-state to its cluster link-state.

A cluster-head node sends a hello message to its cluster members for updating them with the link-state information of other clusters in the network. Every cluster-head that has received one or more cluster link-state unicast messages from other cluster-heads, broadcasts them to its dependents through the hello message along with its own node link-state information. The message format of the hello message sent by a cluster-head node is shown in Fig. 3(a).

As we mentioned earlier, cluster link-state is periodically propagated over the network. It is supposed to be less frequent than hello message. In MCLSR, the rate of hello message is t_c multiple of the rate of cluster link-state propagation. Thus, once every t_c times of hello message, cluster link-state is issued by a cluster-head. At the time, extended hello message is issued, substituting for basic hello message. The structure of extended hello message is also shown in Fig. 3(a). In an extended hello message, there is an additional field called the cluster-head list, where the master cluster-head includes a list of known cluster-heads, which helps cluster-head discovery, as explained later.

Inter–cluster-head unicast message: Once every t_c times of hello message, Inter–cluster-head unicast messages are sent as well as an extended hello message by a cluster-head node to other cluster-head nodes indicating the cluster link-state. The structure of this message is described in Fig. 3(b). It consists of the cluster link state along with the list of known cluster-heads, like an extended hello message. Optionally, a dependent node also can send out an inter–cluster-head message to another cluster-head for cluster-head discovery. In this case, the dependent nodes slice a inter–cluster-head message out of the extended hello message received from its master cluster-head. The detail is described in Section IV-D.

C. Cluster-head selection procedure

The cluster-head selection is performed based on the following rule:

$$\forall v \in V, \exists n \in V_C, \text{ such that } n \in N_l(v) \tag{1}$$
$$\forall m, n \in V_C, m \notin N_t(n) \tag{2}$$

where, N_l and N_t denote the loose and tight neighbors of a node, respectively (see Section III). Among the two equations, Eq. (1) ensures that every node is within a one-hop neighborhood from a cluster-head, and Eq. (2) minimizes the density of cluster-heads by ensuring that no two cluster-heads are tight neighbors of each other. This, in turn, minimizes the number of control packets that are to be broadcast in the network.

In fact, to manage N_l and N_t separately seems to just make the protocol complicate. These two set of neighbors can be defined to be identical and can be called just 'neighbors,' which makes protocol much simpler and easy to understand. However, we distinguish N_l and N_t to reduce control overhead caused by frequent cluster-head reselection. Semantically, Eq. (1) is the condition to promote a cluster-head, and Eq. (2) is the condition to demote a cluster-head. $N_l(v)$ in Eq. (1) makes promotion harder. But once a node is promoted as a cluster-head, it will not be easily demoted because the demotion is based on $N_t(v)$. It prevents frequent change of network configuration, which can make network unstable.

Algorithm 1 describes the cluster-head selection procedure. It is run by each node independently, just before hello message is issued. Here, $id(v)$ is the unique identifier of a node v, which can be the node's IP address (used in our implementation). How to assign unique id (or IP address) is out of the scope of this paper. According to this algorithm, each node checks if there is cluster-head in its neighborhood (Eq. 1). If there is none, then the node itself becomes a cluster-head and generates a unique cluster-head id. If a cluster-head is a tight

Algorithm 1 Decision to be a cluster-head for node v

Require: Collected link-state information
Ensure: Decision to be a cluster-head

```
1: if $v \in V_C$ then
2:     $N(v) \Leftarrow N_t(v)$
3: else
4:     $N(v) \Leftarrow N_l(v)$
5: end if
6: if $N(v) \cap V_C = \varnothing$ then
7:     return TRUE
8: else
9:     if $\forall u \in N(v) \cap V_C, id(v) > id(u)$ then
10:        return TRUE
11:    else
12:        return FALSE
13:    end if
14: end if
```

neighbor of another cluster-head, then the cluster-head with smaller id is demoted a dependent (according to Eq. 2). By performing these decisions, no node is left without cluster-head, and no pair of cluster-heads become tight neighbor at a steady state. Additionally, if a node sees multiple cluster-heads in its neighborhood, then it chooses the one to which link quality is the strongest as its master cluster-head.

D. Cluster-head discovery

Once a cluster-head is elected, it has to discover the other cluster-heads in the network for propagating the cluster link-state information by unicasts. Notice that cluster-head discovery is naturally performed with the cluster link-state propagation. A cluster link-state is delivered to designated nodes, which are the cluster-head's dependents (via hello message) and other cluster-heads (via inter–cluster-head message), whichever the master–cluster-head is aware of. If a designated node knows more cluster-heads than the master–cluster-head, it forwards the cluster link-state to those cluster-heads. As a result, a cluster link-state is propagated further than expected. It results in discovery of an unknown cluster-head to the receiving cluster-heads of the forwarding. In this section, we explain how cluster link-state is forwarded further. Those forwardings are called 'cluster-head discovery' because it results in the discovery. The cluster-head discovery is done in two phases: (1) neighbor–cluster-head discovery and (2) non-neighbor–cluster-head discovery.

The following lemma supports the discovery algorithm; If algorithm works in a steady state network, every cluster-head must discover all the other cluster-heads in accordance with the lemma. Notice that the notion that *'The network is connected'* means that network is not partitioned, such that a path between any pair of nodes are available.

Neighbor–cluster-head discovery

Lemma 1: If a graph $G_C = (V_C, E_C)$ is the graph that connects every pair of cluster-heads in three-hops, the following is satisfied: G_C is a connected graph if and only if G is a connected graph.

Proof: If G_C is connected then G is connected for sure because cluster-heads are all connected and their dependents are connected to cluster-heads.

Suppose that G is connected but G_C is not connected as a contradiction. It means that G_C is partitioned. Suppose that G_C has two disconnected group of cluster-heads, then they are at least four hop distant. Then, there should be a node which is not a cluster head and at least two hop distant from any cluster-head. Then, that node cannot have a neighbor cluster-head, but should be a dependent. It is against the protocol. Lemma is proven. ■

From Lemma 1, we have that a network is connected if cluster-heads are connected in G_C. A pair of cluster-heads, u and v are called *neighbor cluster-heads* when $(u, v) \in E_C$. If they are at one-hop from each other, the discovery is trivial as the nodes come to know each other through their respective hello messages. If they are at a two-hop distance, then there exists a node w, such that $w \in N_l(u)$ and $w \in N_l(v)$. Thus, the nodes u and v can listen to w's hello message, which contains the link-states of (w, u) and (w, v). As discussed in Fig. 2, the link information of this hello message contains a 'cluster-head indicator' that indicates whether the node on the other side of a link is a cluster-head or not. Thus, the nodes u and v can learn about each other through w's hello message.

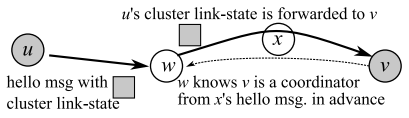

u's cluster link-state is forwarded to v
u
w
x
v
hello msg with cluster link-state
w knows v is a coordinator from x's hello msg. in advance

Fig. 4. Three-hop cluster-head discovery

Discovering a three-hop cluster-head is a little tricky. Fig. 4 shows an example of a three-hop cluster-head discovery. According to this figure, nodes u and v are cluster-heads, and nodes w and x are dependent nodes that lie in between u and v. Let us assume that the cluster-heads u and v do not know about each other to start with. As mentioned before, node u periodically sends out an extended hello message containing cluster link-state information and the list of known cluster-heads to all it dependents. Incidentally, node w becomes aware of the presence of node v within its two hop neighborhood through hello messages sent by node x. Thus, upon receiving the extended hello message from u, the node w compares the list of cluster-heads (contained in the extended hello message) with the list of cluster-heads known to it through hello messages. If there is a mismatch-match between these two lists (which will be the case here as the list sent sent by u will not contain v), the node w prepares an inter–cluster-head message from the extended hello message by retaining the self cluster link-state information and the cluster-head list fields (and chopping off the rest of the fields) and sends it the cluster-heads that are missing in the list sent by u. Thus, in this case, node w will send the inter–cluster-head message to node v. This way, node v can discover that u is a cluster-head (from the self cluster link-state information). Similarly, node u can learn about node v through an inter–cluster-head message from node x or through a subsequent inter–cluster-head message from node v. Because neighbor cluster-heads are cluster-heads in three-hops, all neighbor cluster-heads discover each other, consequently.

Non-neighbor–cluster-head discovery

The procedure for non-neighbor–cluster-head discovery is performed similar to that of neighbor–cluster-head discovery. Once a cluster-head receives an inter–cluster-head unicast message, it parses the cluster-head list to look for any known cluster-heads that is not contained in the received list. If it finds known cluster-heads that is not contained in the list, it forwards the received inter–cluster-head message to the missing cluster-heads immediately, which in turn can learn about the original cluster-head that sent the inter–cluster-head message. The procedure for forwarding the inter–cluster-head message to the missing cluster-heads is discussed in Algorithm 2. Once each undiscovered pair of cluster-heads exchange their cluster link-states, they will be aware of each other; discovery is done.

By neighbor–cluster-head discovery, all neighbor cluster-heads are discovered. By non-neighbor–cluster-head discovery, link-states are forwarded to unknown cluster-heads, which results cluster-head discovery for the receiver of the forwarded link-state. As a result, in a connected network, all cluster-heads are discovered through the link-state forwarding procedure by Lemma 1.

Algorithm 2 Basic link-state propagation to undiscovered cluster-heads

```
Ensure: Decision to be a cluster-head
Require: Inter–cluster-head message $M$ is arrived.
1: $M := [L, V_C^{(\text{msg})}]$
2: $L$: delivered cluster link-state
3: $V_C^{(\text{msg})}$ : Cluster-head list in inter-coord. msg.
4: $V_C^{(\text{rx})}$: known cluster-heads of the receiver
5: $V_C^{(\text{uncovered})} \Leftarrow V_C^{(\text{msg})} \setminus V_C^{(\text{rx})}$
6: if $V_C^{(\text{uncovered})} \neq \varnothing$ then
7:     $M_{\text{new}} \Leftarrow [L, V_C^{(\text{rx})} \cup V_C^{(\text{uncovered})}]$
8:     send $M_{\text{new}}$ to the nodes $\in V_C^{(\text{uncovered})}$ by unicast
9: end if
```

V. MCLSR Implementation

A. Test bed Platform

We implemented MCLSR on top of IEEE 802.11a protocol. IEEE 802.11a has 12 orthogonal channels in 5 GHz band. In practice, at most 5 or 6 channels can be concurrently used with the off-the-shelf devices in accordance with our measurements. The protocol is implemented on Linux 2.4.6 kernel. The main MCLSR protocol runs as a service daemon, and system-oriented parts, like hooking packets and combining two radio interfaces, are embedded in the kernel, as kernel modules. Madwifi driver was used for IEEE 802.11 device driver and modified to obtain short channel-switching time. Basic software architecture is borrowed from Net-X system, so the readers who are highly interested in the software architecture may refer to [8].

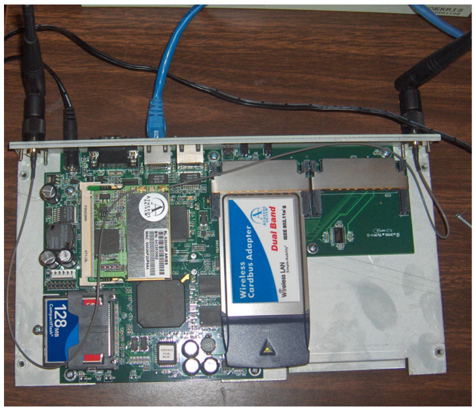


Fig. 5. A picture of test bed

B. Performance Measurements

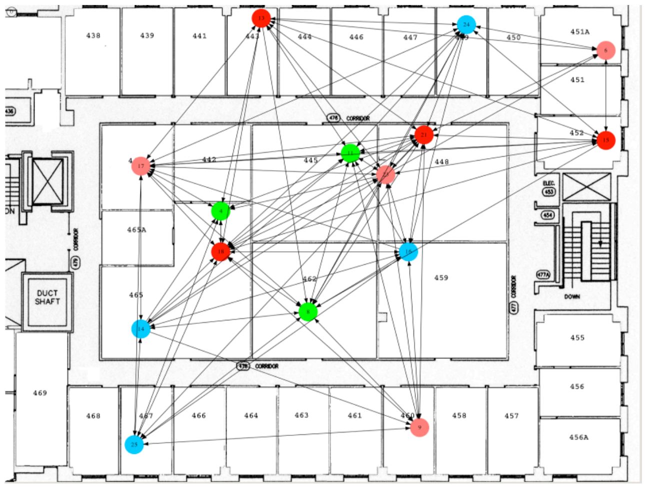


Fig. 6. A deployed node for experiments

a) Route discovery time: Military networks are time- and mission-critical communication environment. Time required for a route discovery is an important metric in military networks. We have measured route discovery time of MCLSR and the reactive protocol. To measure route discovery time, fifteen Soekris boxes (Fig. 5) were deployed on the fourth floor of Coordinated Science Laboratory building of University of Illinois, running either MCLSR or Net-X reactive routing protocol. Each box has two Atheros 802.11a interfaces as a fixed and a switchable interfaces. Fig. 6 depicts our monitoring program, which displays 15 running nodes in the building.

The source node in the route discovery time measurement is Node 6 in Fig. 6, which is the node at the top-right corner.

It constructed routes to all other nodes and measured the packet turn-around time one by one. Fig. 7 shows trivial results that MCLSR has much shorter route discovery time than reactive routing. Because reactive routing has to discover a route using broadcast flooding, it requires certain amount of waiting time to gather enough information in order to build efficient routes. However, MCLSR is ready for transmitting a packet beforehand.

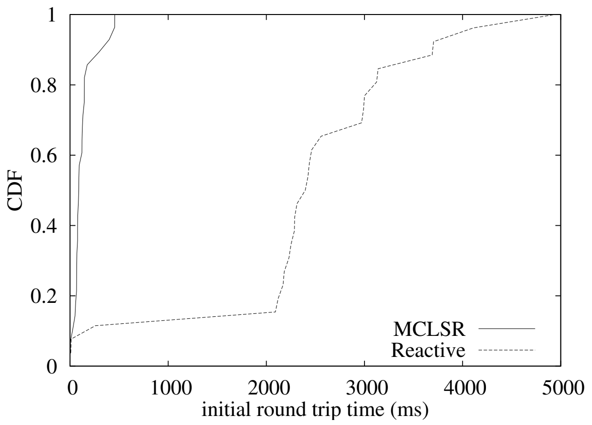


Fig. 7. CDF of packet turn-around time to measure route discovery time for MCLSR and reactive routing.

b) Goodput comparison: We also measured average end-to-end goodput in a random communication scenario. At this time, 23 nodes were deployed in a building, 8 more than the previous experiment, making maximum four hop routes, which is quite similar to Fig. 6. Three communication channels are selected and used for the measurement. In this experiment, every node initiated UDP traffic at a random time to a random destination for a certain amount of time in a certain amount of data rate to realize random communications. And, average end-to-end goodput was measured.

The detail of the experiment is as follows. Before experiments, every node chose communication starting time between 0 and 90 seconds at random. And, it also selected a target node uniformly at random among all the other nodes. Traffic was generated from the chosen starting time to the selected target. Traffic from a source node was sustained for a given time, called *communication duration*, which varies on each run from 2 s to 10 s, and then the source node stopped generating traffic. The data rate per flow also varies on each run, which is either 0.8 Mbps or 1.6 Mbps in our experiment. UDP packet size was 1024 bytes. Every experiment used the same trace of a packet generation pattern to make the comparison fair enough. The experiment ran with either reactive routing or MCLSR.

To capture per-flow goodput, the target node returned acknowledgement packets at the last 20 packets of the traffic, which contained the amount of the delivered data. The source node divides delivered data by the duration from the first packet transmission to the last ack reception, to calculate the goodput. Fig. 8 shows average per-route goodput with respect to communication duration per flow. For each communication duration and routing protocol, five runs were performed and the performance was averaged. When each communication is alive shortly, MCLSR performs better because the reactive routing needs route discovery overhead, which consumes time and communication resources. As communication residence time goes longer, the performance gap between MCLSR and the reactive routing become closer. Notice that many military communications are time-critical and bursty based on strategic decision. The result shows that link-state routing such as MCLSR is more suitable for military networks.

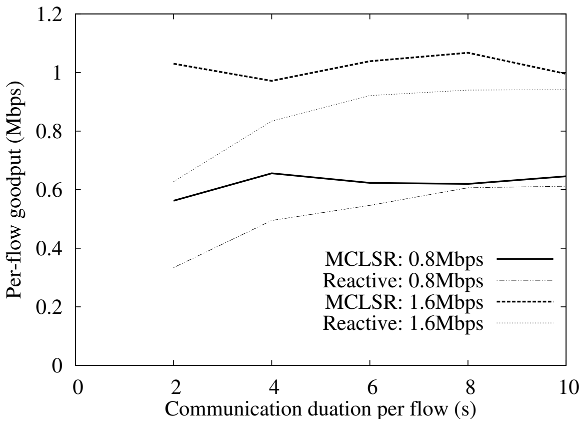


Fig. 8. Average per-route goodput when for MCLSR and reactive routing with respect to communication duration per flow.

VI. Conclusion

Multi-Channel Link-State Routing (MCLSR) protocol for time-critical applications, like military networks, is designed and proposed. It is tailored for multi-channel multi-interface wireless networks using the idea of cluster-heads to minimize broadcast overhead in multi-channel networks. The protocol was implemented on top of IEEE 802.11a test beds, and the performance was measured. The result shows that MCLSR provides transient route discovery time and lower packet drop rate, which is important in mission-critical networking.

References

[1] R. Draves, J. Padhye, and B. Zill, "Routing in Multi-Radio, Multi-Hop Wireless Mesh Networks," in *Proc. of ACM MobiCom*, 2004.

[2] Y. Yang, J. Wang, and R. Kravets, "Designing Routing Metrics for Mesh Networks," in *Proc. of IEEE WiMesh*, 2005.

[3] P. Kyasanur and N. H. Vaidya, "Routing and Link-layer Protocols for Multi-Channel Multi-Interface Ad Hoc Wireless Networks," *SIGMOBILE MC2R*, vol. 10, no. 1, Jan. 2006.

[4] P. Jacquet, P. Mũhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized Link State Routing Protocol for Ad Hoc Networks," in *Proc. of IEEE INMIC*, 2001.

[5] P. A. Spagnolo and T. R. Handerson, "Comparison of Proposed OSPF MANET Extensions," in *Proc. of MILCOM*, 2006.

[6] R. Sivakumar, P. Sinha, and V. Bharghavan, "CEDAR: A Core-Extraction Distrubited Ad Hoc Routing Algorithm," *IEEE Journal On Selected Areas in Communications*, vol. 17, no. 8, pp. 1454–1465, Aug. 1999.

[7] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," *The Springer International Series in Engineering and Computer Science*, vol. 353, pp. 153–181, 1996.

[8] C. Chereddi, P. Kyasanur, and N. H. Vaidya, "Net-x: a multichannel multi-interface wireless mesh implementation," in *Proc. of ACM REALMAN*, 2006.