Extended Abstract: Self-Healing Multi-Radio Wireless Mesh Networks

Kyu-Han Kim and Kang G. Shin
Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan, Ann Arbor, MI 48109-2121, U.S.A.
{kyuhkim, kgshin}@eecs.umich.edu

ABSTRACT

We present novel *L*ocalized s*E*lf-reconfi*G*uration alg*O*rithms (LEGO) for a multi-radio wireless mesh network to autonomously and effectively recover from wireless link failures. First, LEGO locally detects link failures by accurately monitoring the network condition and, upon detection of a failure, triggers network reconfiguration. Second, it dynamically forms/deforms a local group for cooperative network reconfiguration among local mesh routers in a fully-distributed manner. Next, LEGO intelligently generates a local network reconfiguration plan through which a thus-formed group recovers from local failures, while keeping network changes to minimum. Finally, by figuring local channel utilization and reconfiguration cost in its planning, LEGO maximizes the network's ability to meet diverse links' QoS demands. LEGO has been implemented on a Linux-based system and experimented on a real-life testbed, demonstrating its effectiveness in recovering from link failures and its improvement of channel efficiency by up to 92%.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Self-healing networks, link failures, multi-radio wireless mesh network.

1. INTRODUCTION

Despite the increasing abundance of network resources, multi-radio wireless mesh networks (WMNs) still suffer from poor resource utilization and performance degradation due mainly to heterogeneous and fluctuating channel conditions [1, 2]. For example, WMNs in some areas might experience severe channel interference from other co-existing networks, might suffer from mismatched resource allocations due to varying links' QoS requirements, and might not be able to access some frequency bands owing to spectrum regulation in a certain area or during a certain time period [3].

These channel-related problems of WMNs have been researched extensively, but the resultant solutions still suffer from several important problems. First, existing network configuration algorithms [4–6] provide (theoretical) guidelines for network planning, but they usually require "global" changes in network settings, thus incurring high overhead and limiting their scalability. Moreover, they often assume *a priori* availability of accurate information on the network condition and rely on static or periodic network configurations, which are unsuitable for dynamically-changing networks. Second, a *greedy* channel-assignment algorithm [7] can reduce the requirement of network changes for failure recovery, but one local change might cause QoS degradation/failures at neighboring nodes, triggering "propagation" of QoS failures. Finally, fault-tolerant routing protocols such as local re-routing [8] or multi-path routing [9] can be adopted to avoid these side effects. However, their reliance on detour paths or redundant transmissions requires more network resources than network reconfiguration.

To overcome the above limitations, we propose novel *L*ocalized s*E*lf-reconfi*G*uration alg*O*rithms (LEGO) that enable a multi-radio WMN to autonomously reconfigure its settings (e.g., channel and link association) to recover from local link failures. LEGO is a distributed system that is composed of three core *bricks*: monitoring, reconfiguration, and planning. First, the monitoring brick accurately monitors the network condition and detects network failures in real time. The brick in every node efficiently measures the quality of link—such as packet-delivery ratio and data-transmission rate—to each neighboring node by interacting with the network and MAC layers. Further, by adopting hybrid measurement schemes called EAR [10] across multiple radios, LEGO improves both accuracy and efficiency in network monitoring. Based on these measurements, LEGO quickly detects local network failures (e.g., link, QoS, and spectrum failures), triggering network reconfiguration immediately.

Second, the reconfiguration brick forms and deforms a local reconfiguration group, allowing for *on-demand* cooperation among local mesh nodes. Upon receiving a local failure alarm, this brick identifies failure-affected nodes and elects the (local) group leader among the nodes, while collecting necessary information. This information is delivered to the elected leader as part of the group-formation process and is used for reconfiguration planning. Besides the group formation, the reconfiguration brick also includes a deformation protocol that effectively reconfigures network settings based on the planning.

Finally, the planning brick generates the most effective reconfiguration plan that requires only *local* network changes for failure recovery and meets links' QoS requirements. To this end, LEGO takes a top-down approach in its reconfiguration planning. Based on the collected network information, the planning brick in the leader node first enumerates feasible network changes for recovery from detected failures by using constraint graphs. By accepting current network settings as constraints, the brick minimizes

MobiCom'07, September 9–14, 2007, Montréal, Québec, Canada.
ACM 978-1-59593-681-3/07/0009.

changes of healthy network settings, while allowing for local changes around the failure location. Then, the brick selects plans that satisfy links' QoS demands and avoid cascaded network changes, called the *ripple effect*. Finally, of the selected plans, it chooses the most effective plan that improves overall channel utilization.

Our evaluation results on a Linux-based implementation of LEGO show that it outperforms existing failure-recovery methods, such as greedy, local re-routing, and static channel assignments. First, LEGO detects local failures in real time and reconfigures the network, thus improving channel efficiency[1] by up to 92% over the local re-routing protocol. Next, LEGO's planning algorithm effectively generates and identifies reconfiguration plans that improve overall channel utilization and avoid ripple effects.

2. DESIGN OF LEGO

This section outlines the design of LEGO. We first present a design overview, and then describe of each brick of LEGO.

2.1 Overview of LEGO

LEGO is a distributed system that is easily deployable in an existing IEEE 802.11-based multi-radio WMN (e.g., Figure 1) to support localized self-configuration via the following distinct features.

- *In-network monitoring and detection*: LEGO adopts an efficient and accurate link-quality monitor [10] and extends it for a multi-radio WMN. Based on network measurements and given links' QoS constraints, each node detects local link failures, as opposed to relying on a remote server, and, upon detection of a failure, triggers network reconfiguration in real time.
- *Distributed reconfiguration*: LEGO includes a distributed group formation/deformation protocol that helps a WMN reconfigure its settings autonomously. This protocol enables cooperative network reconfiguration among failure-affected mesh routers and minimizes network disruption.
- *Network planning via a constraint graph*: In its reconfiguration planning, LEGO uses a constraint graph to keep network changes as local as possible. By taking current network settings as constraints, LEGO can generate a set of feasible reconfiguration plans that allow network configurations to be changed only in the vicinity of network failures, while retaining configurations in areas remote from failure locations.
- *Estimation of reconfiguration effects*: LEGO effectively identifies QoS-aware and cost-effective reconfiguration plans by (i) estimating an expected bandwidth function for LEGO to use to filter out reconfiguration plans that do not satisfy the QoS requirements; and (ii) deriving channel utilization and reconfiguration cost for evaluation of the benefits and penalty of reconfiguration plans, respectively.

LEGO is composed of three core *bricks* as shown in Figure 2, and their interactions can be described in four procedures: (1) the monitoring brick in every node constantly monitors wireless link-quality and checks each link for possible failures; (2) on detection of a failure, the reconfiguration brick triggers the formation of a local group with the election of a leader; (3) based on collected network information during the group formation, the planning brick in the elected leader node runs a network planning algorithm, generating the most cost-effective plan, and announces the plan to the group members; and (4) the reconfiguration bricks in group members cooperatively reconfigure their NICs and routing settings based on the delivered plan.

[1]Channel efficiency is defined as achieved throughput per unit air-time.

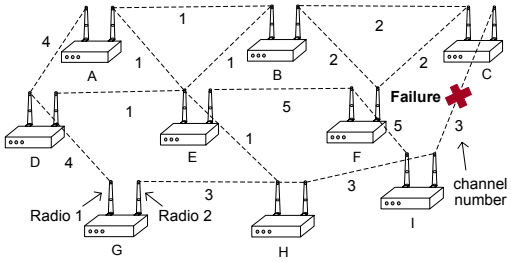

Figure 1: *Multi-radio wireless mesh network*: A WMN has an initial assignment of frequency channels as shown above, but often experiences link failures on a specific channel.

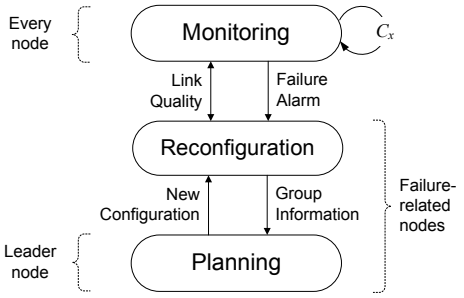

Figure 2: *Interaction diagram among LEGO bricks*: LEGO is composed of monitoring, reconfiguration and planning bricks, running in every node. Each brick is selectively triggered and interacts with each other within/among nodes.

2.2 Network Monitoring

The monitoring brick in LEGO plays a crucial role in autonomous and QoS-aware network reconfiguration. To build such a brick, we must overcome the following challenges: (i) how to accurately measure network information across multiple radios and (ii) how to effectively determine every link's health within the network. These two challenges are elaborated next.

LEGO employs techniques proposed in EAR [10] and apply them to measure link-quality in a multi-radio WMN. LEGO improves measurement efficiency by monitoring existing data traffic, while maximizing accuracy by deriving link-quality from data frame transmission results. In addition, by maintaining a link-state table in the network layer that coordinates measurements across multiple radios, the brick can accommodate as many radios as needed for link-quality monitoring.

The monitoring brick is flexible to detect various types of network failures by employing general parameters such as bandwidth or transmission cost in the model. Using these parameters, this brick periodically checks the health of each link in every node, and triggers an alarm if the link does not satisfy given constraints. Note, however, that this paper focuses on detecting channel-related failures as explained in Section 1, and detecting hardware failures or security attacks are beyond the scope of this paper.

2.3 Distributed Reconfiguration

The reconfiguration brick enables a WMN to reconfigure its settings autonomously and cooperatively among local mesh routers. To this end, the following design issues must be resolved: (i) which and how to form a cooperative reconfiguration group; and (ii) how to reconfigure NIC settings with minimum disruptions.

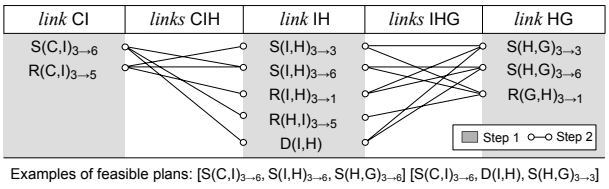

Figure 3: *Example of network planning*: LEGO generates per-link plans (odd columns) and then combines them for feasible reconfiguration plans (even columns) for the multi-radio network with 6 available channels in Figure 1.

On detection of a failure on a specific channel, the reconfiguration brick in a node(s) requests the formation of a reconfiguration group among nodes that use the faulty channel. This formed group is essentially a set of nodes that might be affected by a current link failure. Since the settings of a faulty link must be changed, the neighboring nodes that use the same faulty channel can be affected due to its association.

Once a reconfiguration plan is determined, LEGO deforms group members' network settings and remove a formed group. To minimize traffic disruption during reconfiguration, LEGO uses either asynchronous or synchronous network deformation. LEGO sets a detour path before reconfiguration and then reconfigures networks asynchronously within a time window. If no detour paths exists, it relies on accurate network time protocol (NTP) [11] and changes network settings synchronously.

2.4 Network Reconfiguration Planning

The planning brick in LEGO generates reconfiguration plans for each group. A key idea behind LEGO is that it takes a *top-down* approach to generating reconfiguration plans mainly for minimizing network changes, as opposed to a *bottom-up* approach used in existing scheduling and assignment algorithms [4, 5, 12]. These existing algorithms try to find *one* optimal plan that satisfies the QoS demands on all links in a greedy fashion, and thus, one change in assignments might cause cascaded changes in assignments of other parts. By contrast, LEGO first generates a *set* of feasible local network changes that satisfy current network connectivity. Then, it chooses a plan that satisfies the QoS demand and that improves channel utilization.

To realize such algorithms, the following questions should be answered: (i) how to generate feasible plans, (ii) how to evaluate QoS satisfiability, and (iii) how to identify the most effective plan.

First, LEGO generates a set of feasible plans by enumerating possible changes in each link of a thus-formed group. In this enumeration, LEGO follows a two-step procedure based on the divide-and-conquer principle and uses a *constraint graph* to avoid unnecessary enumerations. As shown in Figure 3, LEGO first generates all possible changes—such as channel switching (`S`), radio association change (`R`), and detouring (`D`)—per each link in a group, and then makes a set of combinations with the possible changes for all links in the group (called feasible reconfiguration plans).

Next, LEGO evaluates each feasible plan with respect to QoS satisfiability and ripple effects. Even though each feasible plan ensures that a faulty link has different network settings, it might not satisfy QoS requirements or even cause cascaded changes of network settings. To filter out such plans, LEGO first estimates the QoS satisfiability of each plan's new network settings, based on measurement results from the monitoring brick. Then, once all links under one plan satisfy QoS requirements, LEGO checks QoS satisfiability up to two-hops-away nodes to identify any ripple effect from a local change(s) and removes plans with the effects.

Finally, LEGO now has a set of reconfiguration plans that are QoS-satisfiable without causing the ripple effect, and needs to choose the best plan among them. To this end, LEGO first calculates average residual bandwidth of links for each feasible reconfiguration plan. Then, it chooses a plan that allows local networks to have evenly distributed residual bandwidth. Finally, if multiple plans have the same distribution, LEGO chooses one plan that incurs the minimum links' changes.

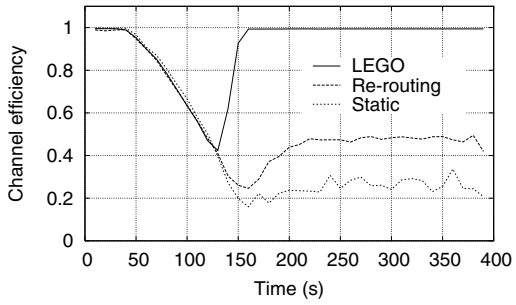

Figure 4: *Gains in channel efficiency:* LEGO effectively reconfigures the network around a faulty link, improving channel efficiency by up to 92%. By contrast, local re-routing causes degradation in channel efficiency, and static channel-assignment does not react to the link failure in a timely manner.

3. PERFORMANCE EVALUATION

LEGO has been implemented in a Linux-based system and evaluated on our testbed. We first describe our experimentation setup, and then present its representative evaluation results.

3.1 Experimental Setup

LEGO's implementation is evaluated in our in-door multi-radio wireless mesh network, constructed in the fourth floor of the Computer Science Building at the University of Michigan. This network is composed of 17 mesh nodes, each of which is a small-size wireless router—Soekris board 4826-50 [13]. Every node is equipped with two EMP IEEE 802.11 a/b/g miniPCI cards and a 5-dBi gain indoor omni-directional antenna, and runs on Linux OS (kernel 2.6) with a modular implementation of LEGO.

3.2 Experimental Results

We now present two sets of experimentation results in the above setup: channel efficiency gain and ripple effect avoidance.

3.2.1 Gain in Channel Efficiency

To show LEGO's gain in channel efficiency—defined as the ratio of the number of successfully-delivered data packets to the number of total MAC frame transmissions, we ran one UDP flow at a maximum rate over links in our testbed, while increasing the level of interference every 10 seconds. We also set the QoS requirements of a link to 6 Mbps as a parameter of LEGO, and measure the channel efficiency progression every 10 seconds during a 400-second run. For the purpose of comparison, we also ran the same scenario under the local re-routing and static channel-assignment algorithms.

As shown in Figure 4, since LEGO uses a better-quality channel than the one on the faulty link via real-time reconfiguration, the UDP flow can achieve a high successful frame transmission ratio, improving channel efficiency by up to 91.5%. On the other hand,

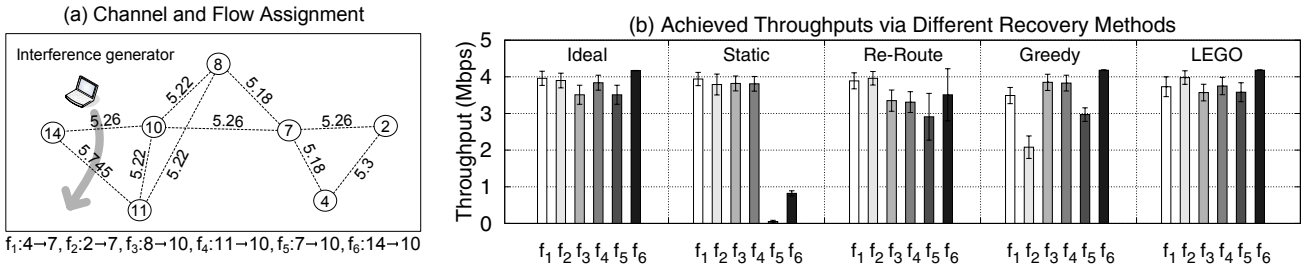

Figure 5: *LEGO's avoidance of ripple effects*: LEGO finds a local reconfiguration plan that avoids the ripple effects by considering neighboring nodes' channel utilization, whereas the greedy channel switching and local re-routing triggers additional QoS failures in neighbors' flows, as shown in Figure 5(b).

using static-channel assignment suffers a poor packet-delivery ratio, resulting in severe degradation of channel efficiency. Finally, local re-routing often makes traffic routed over longer and low link-quality paths, thus consuming more channel resources to deliver the same amount of traffic than LEGO. Note that we do not intentionally run a greedy algorithm in this single-hop scenario, because its gain is the same as LEGO. We, however, compare it with LEGO in multi-hop scenarios next.

3.2.2 Avoidance of Ripple Effects

We also studied LEGO's effectiveness in reducing the ripple effects of network reconfiguration. Figure 5(a) shows initial channel and flow assignments in our testbed. In this topology, we ran 6 UDP flows ($f_1, \cdots, f_6$) each at the rate of 4Mbps, and measure each flow's throughput while injecting interference into a target channel. Also, we use the four reconfiguration methods (i.e., static, local re-routing, greedy, and LEGO) for failure recovery and measure the effects of each method on reconfiguration. We ran the above scenario with interference in 5.28 Ghz to induce failures of links that use channel 5.26 Ghz. Finally, we also measure the throughput of each flow without any interference for the *ideal* case.

Since LEGO considers the effects of local changes on neighboring nodes in its planning, it reconfigures the network settings to effectively avoid these adverse effects on other nodes. Figure 5(b) shows the average throughput of each flow after network reconfiguration. First, with interference in 5.28 Ghz, links among nodes 14, 10 and 7 experience link-quality degradation. By using LEGO, each flow achieves an average 98% of the ideal throughput after reconfiguration, whereas local re-routing achieves 82% of the throughput because of its use of detour paths. On the other hand, the greedy approach causes degradation of neighboring links, while partially recovering from the original link failures. This is because one local greedy channel switching (from 5.26 to 5.32 Ghz) requires the neighboring links' channel (e.g., link between nodes 7 and 2) to change, creating interference to other neighboring nodes' link (e.g., link between nodes 2 and 4) that use adjacent channels.

4. SUMMARY

We presented localized self-reconfiguration algorithms (LEGO) for a multi-radio wireless mesh network to autonomously recover from frequent local link failures. The monitoring brick in LEGO enables on-line network reconfiguration through an efficient and accurate network monitoring and real-time failure detection. The reconfiguration brick allows cooperative reconfiguration among local mesh routers during the failure recovery. Finally, LEGO's planning brick generates a cost-effective reconfiguration plan that recovers from link failures and avoids adverse ripple effects.

LEGO has been implemented on a Linux-based system and evaluated on a real-life testbed, demonstrating its improvement of channel efficiency by up to 92% and the effectiveness in avoiding ripple effects. Our future work includes a joint optimization of channel and flow assignments, and measurement study for link-failure characterization in multi-radio WMNs.

5. REFERENCES

[1] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-level measurements from an 802.11b mesh network," in *Proceedings of ACM SIGCOMM*, Portland, OR, Aug. 2004.

[2] A. Akella, G. Judd, S. Seshan, and P. Steenkiste, "Self-management in chaotic wireless deployments," in *Proceedings of ACM MobiCom*, Cologne, Germany, Sept. 2005.

[3] M. J. Marcus, "Real time spectrum markets and interruptible spectrum: New concepts of spectrum use enabled by cognitive radio," in *Proceedings of IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks*, Baltimore, MD, Nov. 2005.

[4] M. Alicherry, R. Bhatia, and L. Li, "Joint channel assignment and routing for throughput optimization in multi-radio wireless mesh networks," in *Proceedings of ACM MobiCom*, Cologne, Germany, Aug. 2005.

[5] M. Kodialam and T. Nandagopal, "Characterizing the capacity region in multi-radio multi-channel wireless mesh networks," in *Proceedings of ACM MobiCom*, Cologne, Germany, Aug. 2005.

[6] A. Brzezinski, G. Zussman, and E. Modiano, "Enabling distributed throughput maximization in wireless mesh networks-a partitioning approach," in *Proceedings of ACM MobiCom*, Los Angeles, CA, Sept. 2006.

[7] A. Raniwala and T. Chiueh, "Architecture and algorithms for an IEEE 802.11-based multi-channel wireless mesh network," in *Proceedings of IEEE InfoCom*, Miami, FL, Mar. 2005.

[8] S. Nelakuditi, S. Lee, Y. Yu, J. Wang, Z. Zhong, G. Lu, and Z. Zhang, "Blacklist-aided forwarding in static multihop wireless networks," in *Proceedings of IEEE SECON*, Santa Clara, CA, Sept. 2005.

[9] S. Chen and K. Nahrstedt, "Distributed quality-of-service routing in ad hoc networks," *IEEE JSAC*, vol. 17, no. 8, pp. 1488–1505, 1999.

[10] K.-H. Kim and K. G. Shin, "On accurate measurement of link quality in multi-hop wireless mesh networks," in *Proceedings of ACM MobiCom*, Los Angeles, CA, Sept. 2006.

[11] D. L. Mills, "Network time protocol (version 3)," Internet Request for Comments 1305 (rfc1305.txt), Mar. 1992.

[12] K. Ramanchandran, E. Belding-Royer, and M. Buddhikot, "Interference-aware channel assignment in multi-radio wireless mesh net works," in *Proceedings of IEEE InfoCom*, Barcelona, Spain, Apr. 2006.

[13] Soekris Engineering, http://www.soekris.com.

Acknowledgement

The work reported in this paper was supprted in part by NSF under Grant CNS 0519498 and Intel Corporation.