Routing with Metric-based Topology Investigation

Frank Bohdanowicz, Harald Dickel, and Christoph Steigner
Institute for Computer Science
University of Koblenz-Landau
{bohdan,dickel,steigner}@uni-koblenz.de

ABSTRACT

As routing takes place in an entirely distributed system where local routers have no direct access to globally consistent network state information, a routing algorithm has to make uncertain forwarding decisions. As the network state may change, due to failures or new adoptions of networks, routing algorithms have to adapt themselves to the new situation. This network convergence phase should be carried out as quickly and precisely as possible. Besides the problem of generating the proper updates for the locally distributed routers, the problem of forwarding the routing updates is also manifest: routing updates travelling along routing loops may become obsolete or outdated. We developed a new distance vector algorithm which solves the problem of routing loops. This provides distance vector routing with crucially improved convergence, stability, and scalability abilities, thus making distance vector routing once again an attractive revitalized alternative to link state routing.

Keywords– routing; distance vector routing; metric-based topology investigation; routing loops; counting to infinity problem; routing convergence

1. INTRODUCTION

The major goal of all routing algorithms is to achieve a fast and correct convergence after a change in the network topology. In this phase, the forwarding of the actual routing updates is crucial in contrast to the forwarding of existent but obsolete update information. In distance vector algorithms, routing updates which have made their way along network loops contain in most of the cases obsolete information which should not be considered anywhere further on. Whenever network topologies in the Internet contain loops, alternative routes are available in case of a link failure. Unfortunately, topology loops complicate the correct detection of routes.

The major approaches to cope with this problem are distance vector, link state, and vector path algorithms.

Distance vector routing algorithms like the Routing Information Protocol (RIP) [10] cannot cope with routing loops efficiently. A routing loop is the trace of a routing update which occurs at a router again reporting seemingly new reachability information which is based on already known information. This event results in a temporary inconsistency during the convergence process. The well-known split horizon approach fails if the network topology contains loops. In this case, the routing convergence is impeded because invalid old routing updates may find their way along topology loops and cause routing loops which appear as the well-known *counting to infinity* (CTI) problem [10]. RIP, as a classic representative of the distance vector protocol family, can only inadequately cope with CTIs by limiting the metric to a small maximum. This does not solve the CTI problem since misguided data traffic may congest the trace of the routing loop and the maximum metric cannot be reached in a short time. Up to now there has been no proper solution for the CTI problem.

We show in this paper that the distance vector approach can be improved by a mechanism that can recognize all those outdated updates which were propagated along loops. Our simulation results and analysis show that distance vector routing can be significanty improved. Our new Routing with Metric-based Topology Investigation (RMTI) protocol can alleviate the CTI problem found in distance vector routing protocols like RIP. Our RMTI protocol is entirely compatible to RIP since it uses the same routing update message format. The convergence time of our RMTI protocol does not depend on an upper metric limit, so it is applicable in large-scale network environments.

Path vector routing like the interdomain Border Gateway Protocol (BGP) [17] was designed to solve the routing loop problem by including the entire path (AS-path) from source to destination in its updates in order to detect the occurrence of routing loops. It has, however, been shown that BGP suffers from forwarding loops during routing convergence after topology changes [12, 15].

Due to the drawbacks of the classical distance vector routing, in recent years the focus in further development of interior routing protocols was on link state routing. But link state routing like Open Shortest Path First (OSPF) [11] do not solve the routing loop problem entirely due to the fact that these routing algorithms also suffer from forwarding loops [6, 8, 22]. The brute force effort of the link state algorithms, the overhead prone reliable flooding technique, limits its deployment [9]. Besides this, the link state approach has some disadvantages such as the absence of local routing policy facilities which provide network administrators with comprehensive capabilities to influence traffic density. In distance vector and path vector routing, the routing update flow is directly related to the actuated traffic flow. Thus distance vector and path vector algorithms can naturally cope with routing policies. By our deployment of RMTI, we show that distance vector routing algorithms can be an attractive alternative to the link state routing suite.

This paper gives a detailed description of our RMTI protocol, implementation, and evaluation results based on [1, 2]. The paper is organized as follows: In Section 2 we give a short overview of other approaches to distance vector routing which solve the CTI problem. In Section 3 we discuss the routing problem as a loop problem and state our vocabulary of loop concepts. In Section 4 we present the principles of our new RMTI approach. In Section 5 we present our protocol together with some characteristic examples of routing loop detection and update rejection. In Section 6 we describe our implementation. We close with our conclusion in Section 7.

2. RELATED WORK

To avoid routing loops and the CTI problem, several enhanced distance vector protocols which increase the amount of information exchanged among nodes and new routing architectures have been proposed.

The Ad hoc On-Demand Distance Vector (AODV) protocol [14] by Perkins expands the distance vector information originally based on subnet N, next hop NH and distance D, to a 4-tuple (N,NH,D,SEQ), where SEQ denotes the sequence number. The result is that although this approach is provably loop free [13], it is not compatible with RIP due to the required protocol changes. The Enhanced Interior Gateway Routing Protocol (EIGRP) used by Cisco is based on the DUAL algorithm [7] proposed by Garcia-Luna-Aceves. DUAL provides loop-free paths at every instance, which was proved in [7]. However, it is a Cisco proprietary routing protocol and not compatible with RIP due to a different protocol design. A solution called Source Tracing was proposed by Cheng et al [3] and Faimann [5]. In this approach, updates and routing tables provide additional information by adding a first-hop indication (the head of the path). Loops can be recognized recursively.

These protocols avoid the CTI problem because they provide loop freedom, but they are likewise not compatible with the RIPv2 standard or are proprietary approaches.

In contrast to these approaches, we aim to provide a solution that has a complete and solid backward compatibility with every existing implementation of RIPv2 [10]. The enhanced knowledge is based on the information already provided by the RIP protocol. So even deploying a new RMTI router only at selected nodes is possible.

3. THE ROUTING PROBLEM

In a computer network, a router's task is to connect several subnets to build an internetwork. Routers have to ensure that, within this internetwork, communication between arbitrary locations in different subnets becomes possible. In the following we use the term *network* as a synonym for the term *internetwork*. The Internet Protocol (IP) is the key communication protocol in such networks. It uses packet forwarding to deliver a data packet addressed to a destination in a certain subnet. A data packet is forwarded from router to router until it reaches a router which is directly connected to the data packet's destination subnet. Finally this router delivers the data packet to its destination. A router has to know which router is the *next hop router* in the forwarding process in order to reach a subnet. Therefore it maintains a forwarding table. Basically a forwarding table is a list of entries containing the next hop router for every subnet. The task of building up and maintaining a correct forwarding table is called *routing*. Usually, routing is achieved by a routing protocol which is running distributed among the routers. The fact that a routing protocol is a distributed system offers some favorable and welcome properties such as enhanced reliability and scalability, when compared to a centralized approach.

A real network is far from being a static entity. New subnets are connected, sometimes old ones are removed, new links between subnets are established via routers, or existing links and routers may fail. A routing protocol has to deal with all these events. Therefore routers have to communicate among each other all changes in the network relevant to the routing task. They are exchanging *update messages* to announce their actual view of the network state. Unfortunately it always needs some time to distribute the update messages all over the network. It is impossible to guarantee that all routers share a common and consistent view at any one time. Furthermore it is possible that *update messages* on their way through the network become outdated. The information enclosed may be no longer valid and may cause misleading assumptions by a router which receives such outdated information. It is still an ongoing challenge to design a routing protocol and make it work as a real distributed system that solves all routing problems.

All these problems get worse with the presence of loops in the network topology. Further on we will look at the problems caused by loops in detail.

Loops appear in different forms and on several occasions throughout this paper. We use five variations of the term *loop*: topology loop, forwarding loop, routing loop, Simple Loop, and Source Loop. To avoid confusion, we have to make a proper distinction between these concepts and the usage of the term loop in this paper.

A *topology loop* is a loop within the network based on the physical network topology. In Figure 1a we have the topology loop $(r_1, s_1, r_2, s_2, r_3, s_3, r_1)$ and in Figure 1c an additional topology loop $(r_1, s_4, r_4, s_5, r_5, s_6, r_1)$. Topology loops add redundancy by offering multiple routes to certain subnets and enhance the reliability of a network.

Data packet delivery in an IP network is done by hop by hop packet forwarding as stated above. A data packet addressed to a destination in a certain subnet s is sent to the appropriate next hop router listed in the forwarding table. If a data packet forwarded by a router r returns to router r again after some intermediate hops, the entries in the forwarding tables of the routers have built a *forwarding loop*. Once a data packet gets into such a forwarding loop, it sticks in this loop and circles around, never reaching its destination.

The functionality of the data forwarding principle is based on the assumption that the next hop router has a shorter distance to the subnet s than the actual router. Unfortunately, different routers may have a different and inconsistent view of the distances to a destination subnet caused by the distributed nature of the routing protocol. This inconsistent view may lead to the configuration of a forwarding loop. Forwarding loops can consume a large amount of network bandwidth and can impact the end-to-end performance of the network. Therefore it is necessary to recognize and to prevent forwarding loops. In practice all common routing protocols suffer from forwarding loops during the convergence process after a topology change in the network [8].

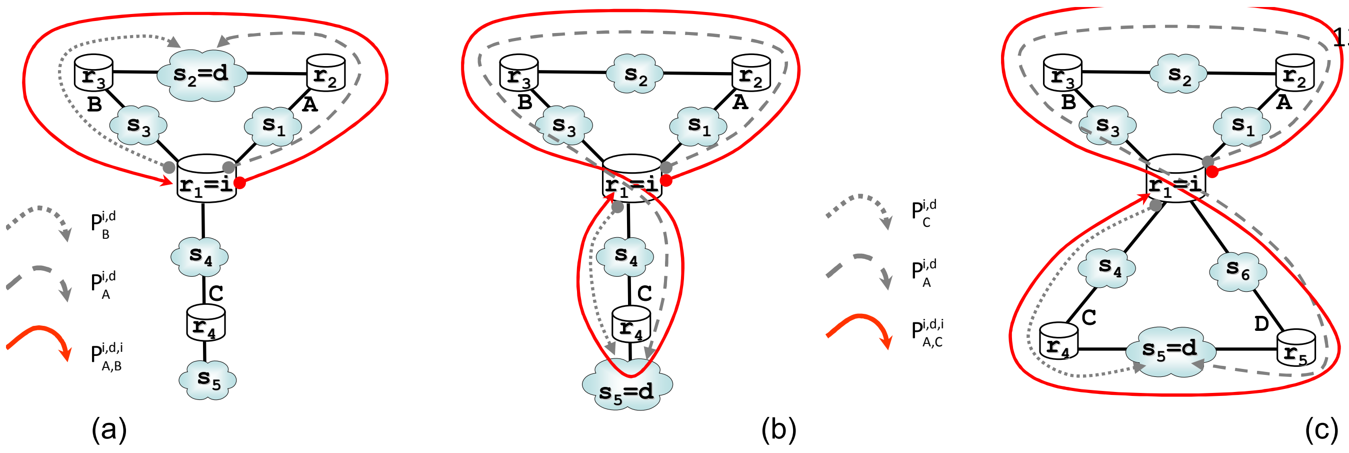


Figure 1: Three examples of loops in a network. (a) shows a Simple Loop $P^{i,d,i}_{A,B}$ between the neighbor interfaces A and B. (b) shows a Source Loop $P^{i,d,i}_{A,C}$ between the neighbor interfaces A and C. The path $P^{i,d}_{B}$ is part of the path $P^{i,d}_{A}$. (c) shows a different topology with another Source Loop $P^{i,d,i}_{A,C}$ between the neighbor interfaces A and C. Here the path $P^{i,d}_{B}$ is not enclosed in the path $P^{i,d}_{A}$.

Forwarding loops arise in link state routing by computing inconsistent shortest path trees by distinct routers on the basis of different link state databases [22], or in distance vector routing due to the occurrence of a routing loop.

While the term *forwarding loop* denotes a loop in the forwarding process of a data packet, we use the term *routing loop* exclusively to denote a loop in the trace of a routing update. Like a forwarding loop reflects a loop in the forwarding process, a routing loop reflects a loop in the routing process. The routing process manages the forwarding table of a router and has direct influence on the forwarding process. Every routing loop is in close relation to a forwarding loop. If router r_1 in Figure 1a sends a routing update designating the reachability of subnet s_4 to router r_2, then r_2 may insert an updated entry to subnet s_4 in its forwarding table. Now r_2 forwards data packets to s_4 via r_1. The traffic flow takes the opposite direction of the update flow. So in fact every routing loop causes a corresponding forwarding loop in the opposite direction.

In distance vector routing, routing loops appear as the counting to infinity problem. Our RMTI approach avoids CTIs by evaluating the metrics of the routing updates more carefully than other distance vector algorithms. RMTI detects and distinguishes two different shapes of loops composed of traces of routing updates. We call these loops *Simple Loop* and *Source Loop*. A Simple Loop (Figure 1a) is a path which leaves a distinct router at one interface and comes back to the same router on another interface, *without having passed* through the same router in between. A Source Loop (Figure 1b+c) is a path which leaves a distinct router at one interface and comes back to the same router on another interface, *having passed* the same router in between.

In contrast to RIP, we can detect these loops by not deleting old routing information as soon as new and better information arrives at a router; rather, we maintain some information in order to detect Simple Loops and Source Loops.

We show that the detection of Source Loops enables us to avoid all kinds of CTI situations in all loop topologies by rejecting malicious routing updates. We implemented a fully functional version of RMTI and did comprehensive tests with different network topologies in order to analyze and evaluate the RMTI behavior.

4. DESIGN RATIONALE

Now we present a formal network model and a notation which is sufficiently comprehensive to sketch out and prove the concept of our approach.

4.1 The Network Model

A computer network is a collection of devices like hosts, routers, switches, or subnets which are connected via network links. In formal modeling, a computer network is typically represented by a graph. The devices are the nodes of the graph and the edges correspond to the network links. The routing problem is to find a path between any two nodes. For the purpose of discussing the routing problem in computer networks, it is sufficient to consider two types of nodes (subnets and routers) and one type of edges (interfaces). Figure 2 shows a network graph with 7 subnets $(s_1,\ldots,s_7)$, 5 routers $(r_1,\ldots,r_5)$ and 13 interfaces $(A,\ldots,M)$. The number of subnets $s_1,\ldots,s_n$ are subsumed to the set $\mathcal{S} = \{s_1,\ldots,s_n\}$. The subnets are connected via routers and the routers are attached to subnets via interfaces. The interfaces represent the network links and correspond to the edges of the graph.

In general we use upper-case characters to denote interfaces and $\mathcal{I}$ is the set of interfaces $\mathcal{I} = \{A,B,C,D,\ldots\}$. An interface is assigned to exactly one router and in our model a router is fully specified by its interfaces. So, we use the interfaces which are assigned to a router r to define $r = \{U_1,U_2,\ldots,U_n\}, U_1\ldots U_n \in \mathcal{I}$. In Figure 2 by example we have $r_1 = \{A,B,C\}, r_2 = \{D,E\},\ldots,$ etc.

Let $\mathcal{R}$ denote the set of routers $\mathcal{R} = \{r_1,\ldots,r_k\}$. Since an interface is an element of one unique router only, we have $\forall U \in \mathcal{I}\ (U \in r_i \Rightarrow U \notin r_j), \quad r_i,r_j \in \mathcal{R}, i \neq j$.

Every interface is attached to exactly one subnet. An interface is the link of a router to one certain subnet.

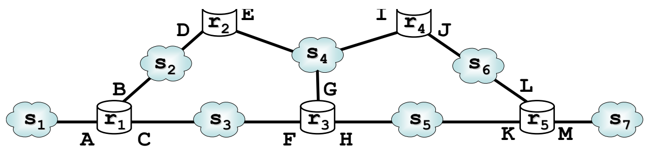


Figure 2: Formal representation of a network.

The topology of the network graph is given by the relation $CON \subseteq \mathcal{I} \times \mathcal{S}$ and defines the mapping between interfaces and subnets: $(U_i, s_j) \in CON, U_i \in \mathcal{I}, s_j \in \mathcal{S}$, if and only if interface U_i is connected to subnet s_j.

Finally, a router $r \in \mathcal{R}$ is connected to subnet $s \in \mathcal{S}$, if and only if one of its interfaces $U \in r$ is connected to subnet s that is $\exists U \in r$ with $(U, s) \in CON$.

Given the sets of subnets $\mathcal{S}$, interfaces $\mathcal{I}$, and routers $\mathcal{R}$ together with the connection relation CON, we have a complete formal specification of the topology of a network. It can be represented as a graph like the one in Figure 2. This graph is the representation of the following specification:
$\mathcal{S} = \{s_1, \ldots, s_7\}, \mathcal{I} = \{A, \ldots, M\}, \mathcal{R} = \{r_1, \ldots, r_5\}$ with $r_1 = \{A, B, C\}, r_2 = \{D, E\}, r_3 = \{F, G, H\}, r_4 = \{I, J\}, r_5 = \{K, L, M\}$, and $CON = \{(A, s_1), (B, s_2), (C, s_3), (D, s_2), (E, s_4), (F, s_3), (G, s_4), (H, s_5), (I, s_4), (J, s_6), (K, s_5), (L, s_6), (M, s_7)\}$

Router nodes are drawn as cylinder, subnet nodes as clouds, and there is an edge between a router $r \in \mathcal{R}$ and a subnet $s \in \mathcal{S}$ if and only if $\exists U \in r$ with $(U, s) \in CON$.

To discuss the basic questions in the area of routing, namely "Is there a route from a certain node in the network to a certain destination?", we have to define some useful terms and properties about transitions from node to node in a network graph.

An elementary step from a router $i \in \mathcal{R}$ via outgoing interface $O \in i$ to an adjacent router $j \in \mathcal{R}$ via incoming interface $I \in j$ using subnet $s \in \mathcal{S}$ is called a *hop* (see Figure 3). It is defined by a 3-tuple $H^{i,j} = (O, s, I)$, whereas $(O, s), (I, s) \in CON$.

If the destination of a hop is simply given by a subnet and not by a designated interface of a router, we use the special symbol $*$ as the last element of the 3-tuple, i.e. the hop from router i to subnet d is notated as $H^{i,d} = (O, d, *)$.

DEFINITION 1. *(Hop)*
A hop H *from a router* $i \in \mathcal{R}$ *via outgoing interface* $O \in i$ *to an adjacent router* $j \in \mathcal{R}$ *via incoming interface* $I \in j$ *using subnet* $s \in \mathcal{S}$ *is the 3-tuple* $H = (O, s, I)$ *with* $(O, s), (I, s) \in CON$ *and* $\mathcal{H}$ *denotes the set of all hops.*

In abbreviated form we use the router identifiers as a superscript and the interface identifiers as a subscript to specify a hop. The notation $H^{i,j}_{O,I}$ indicates that this hop is

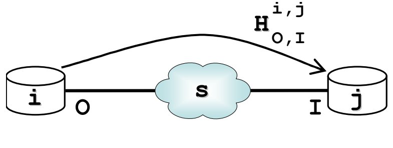


Figure 3: The hop $H^{i,j}_{O,I} = (O, s, I)$ from router i to router j uses the outgoing interface O and reaches the incoming interface I of router j via subnet s.

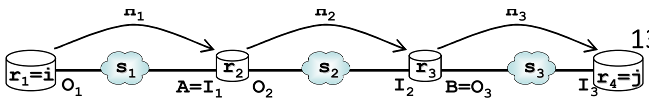


Figure 4: A path $P^{i,j}_{A,B}$ from router i to router j.
$P^{i,j}_{A,B} = (H_1, H_2, H_3) = ((O_1, s_1, A), (O_2, s_2, I_2), (B, s_3, I_3))$

leaving router i via interface O and leading to router j via interface I (Figure 3).

If two routers i and j are connected to the same subnet s, then there exists one hop $H^{i,j}$ from i to j as well as a hop $H^{j,i}$ from j to i. In this case i and j are called *neighbors*.

DEFINITION 2. *(Neighbor)*
Let $i, j \in \mathcal{R}, i \neq j$ *be two distinct routers.* j *is a* neighbor *of* i, *iff* $\exists O \in i, I \in j$, *and* $H^{i,j}_{O,I} \in \mathcal{H}$.

In Figure 2 r_1 is a neighbor of r_2 and r_2 is a neighbor of r_4 but r_1 and r_4 are not neighbors.

If router j is a neighbor of router i there exists a hop $H^{i,j}_{O,I}$ from i to j. According the definition of a hop, O is an interface of router i and I is an interface of i's neighbor j, where O and I are connected to the same subnet s. To point out this relationship between router i and interface I we call I a *neighbor interface* of i. In Figure 2 r_2 has the three neighbor interfaces B, G, and I.

A path through the network is a sequence of hops (Figure 4).

DEFINITION 3. *(Path)*
A path $P^{i,j}_{A,B}$ *beginning at router* $i \in \mathcal{R}$ *leading to router* $j \in \mathcal{R}$ *is a sequence of hops*

$$\begin{aligned} P^{i,j}_{A,B} &= (H_1, H_2, \ldots, H_l) \\ &= ((O_1, s_1, I_1), (O_2, s_2, I_2), \ldots, (O_l, s_l, I_l)) \\ &= ((O_1, s_1, A), (O_2, s_2, I_2), \ldots, (B, s_l, I_l)) \end{aligned}$$

with $O_1 \in i, I_l \in j$, $I_1 = A, O_l = B$ *and* $\exists\ r \in R$ *with* $I_j, O_{j+1} \in r$ *for* $1 \leq j < l$. *The metric of* $P^{i,j}_{A,B}$ *is* $m^{i,j}_{A,B} = l$ *which is simply the number of hops and* $\mathcal{P}$ *denotes the set of all paths.*

Again, we use an abbreviated form with superscripts to indicate the start and the destination router of a path and subscripts to specify the first and the last hop of a path more precisely. By using the notation $P^{i,j}_{A,B}$ of a path $P \in \mathcal{P}$, we specify that interface A is a neighbor interface of router i, and interface B is a neighbor interface of router j. With this notation we can distinguish in Figure 2 a path from r_2 to r_5 via r_3 and a path via r_4. The former is noted $P^{r2,r5}_{G,H}$ and the latter $P^{r2,r5}_{I,J}$.

If a path from router i leads to a subnet d, then the last hop H_l in the sequence of hops takes the form $H_l = (O_l, d, *)$ and we simply write $P^{i,d}$ using $d \in \mathcal{S}$ as a superscript instead of a destination router. And finally, we write $P^{i,d,j}$ to denote a path $P \in \mathcal{P}$ which traverses subnet d. In this case there exists a hop $H_k, 1 \leq k \leq l$ in P with $H_k = (O_k, d, I_k)$.

4.2 Simple Loops and Source Loops

A closed path $P^{i,i}_{A,B}$ begins and ends at router i through the network where the first hop $H_1 = (O_1, s_1, A)$ leaving router i and the last hop $H_l = (B, s_l, I_l)$ returning to router i use the different neighbor interfaces A and B (Figure 1).

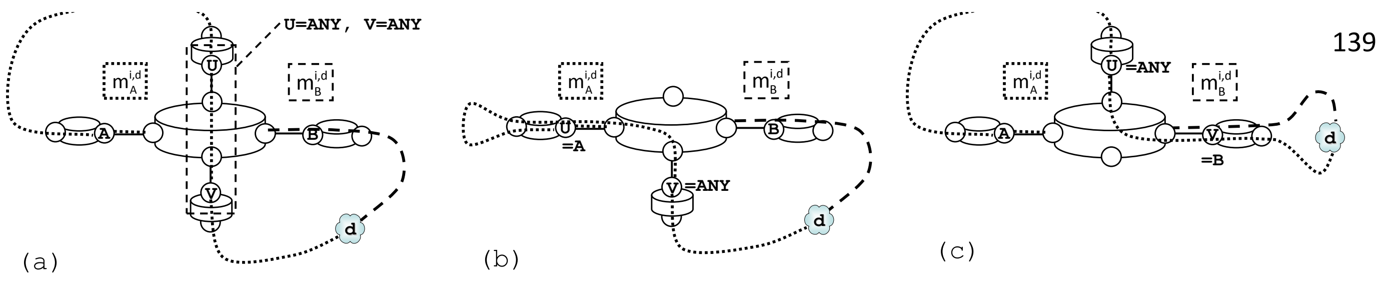


Figure 5: Source Loop types. If a path $P_{A,B}^{i,d,i}$ is a Source Loop, the router i is passed in between via two neighbor interfaces. These may be interface A, B, or ANY (any other) neighbor interface of router i. Therefore we have $3^2 = 9$ possible Source Loop types (Table 1). Let us call these intermediate interfaces U and V. (a) shows the Source Loop type with $U = ANY$ and $V = ANY$, (b) results with $U = A$ and $V = ANY$, and in (c) we have $U = ANY$ and $V = B$.

It is tempting to assume that such a closed path offers two distinct useful paths to a subnet d that is part of this path $P_{A,B}^{i,d,i}$ as shown in Figure 1a. However, this has to be considered very carefully. Figures 1b and 1c show examples that this assumption does not hold in general.

Figure 1a shows the network graph of an example network topology. If we examine paths from router $r_1 = i$ to a destination subnet $d = s_2$, there is a path $P_A^{i,d}$ via interface A from neighbor router r_2 and a path $P_B^{i,d}$ via interface B from neighbor router r_3 to subnet d. If we combine these paths, we obtain a closed path $P_{A,B}^{i,d,i}$, starting at router i via neighbor r_2 and ending at router i coming in from neighbor r_3.

The network topologies of Figures 1b and 1c show a different situation. Again there are two different paths from router $r_1 = i$ to a destination subnet d. There is a path $P_C^{i,d}$ from router i via neighbor interface C to subnet $d = s_5$ and a path $P_A^{i,d}$ from router i via neighbor interface A to subnet d. If we combine these paths, we obtain a closed path $P_{A,C}^{i,d,i}$ once again with both paths building a loop. But there is an important difference between the path $P_{A,B}^{i,d,i}$ in Figure 1a and the two paths $P_{A,C}^{i,d,i}$ in Figures 1b and 1c.

In Figure 1a there is a topology loop between the neighbor interfaces A and B of router i. If the path $P_B^{i,d}$ is no longer usable (due to a link or router failure on that path), there is an alternative path $P_A^{i,d}$ to subnet d which router i might use. But in the topologies of Figures 1b and 1c the paths $P_{A,C}^{i,d,i}$ traverse router i in between. There is no topology loop via the neighbor interfaces A and C at router i. If the path $P_C^{i,d}$ is no longer usable, there is no alternative path $P_A^{i,d}$ to subnet d which router i might use since $P_C^{i,d}$ is part of the path $P_A^{i,d}$. Therefore we have to distinguish between these two different types of closed paths. A path from router i, traversing a subnet d, ending at router i but never passing router i in between is called a Simple Loop.

Definition 4. *(Simple Loop)*
A Simple Loop is a path $P_{A,B}^{i,d,i}$ where $O_1, I_l \in i$, $I_1 = A$, $O_l = B$, $\exists n\ 1 \leq n \leq l\ s_n = d$, and $\forall I_j\ 1 \leq j < l\ I_j \notin i$.

We denote the set of all Simple Loops within a network with SIL and the metric of a Simple Loop $P_{A,B}^{i,d,i} \in SIL$ is $\text{silm}_{A,B}^{i,d,i} = m_{A,B}^{i,d,i} = l$.

A path from router i, traversing a subnet d and ending at router i but this time passing router i in between is called a Source Loop.

Definition 5. *(Source Loop)*
A Source Loop is a path $P_{A,B}^{i,d,i}$ where $O_1, I_l \in i$, $I_1 = A$, $O_l = B$, $\exists n\ 1 \leq n \leq l\ s_n = d$, and $\exists I_j\ 1 \leq j < l\ I_j \in i$

Figure 5 explains the possible Source Loop types depending on what interfaces are used by the intermediate passing of router i. Only a Simple Loop provides an alternative path to a destination subnet (Figure 1a). On the other hand, the attempt to use the first hop of a Source Loop $P_{A,B}^{i,d,i}$ in order to reach subnet d results in the configuration of a routing loop (Figure 1b and 1c). Discovering a Simple Loop at a router requires the exclusion of the possibility that a closed path $P_{A,B}^{i,d,i}$ is a Source Loop. In order to detect Source Loops, we need to identify the Simple Loop with the lowest metric out of a set of recognized Simple Loops of different metric sizes. This can be done by simply inspecting all Simple Loop metrics.

The minimal Simple Loop metric (msilm) between two neighbor interfaces A and B on router i is:

$$\text{msilm}_{A,B}^{i} = \min\{\text{silm}_{A,B}^{i,d,i} \text{ for all subnets } d\}$$

Furthermore, we define the minimal return path metric (mrpm) which can be derived from the minimal Simple Loop metric (msilm). The minimal return path metric (mrpm) via an interface A on router i is:

$$\text{mrpm}_{A}^{i} = \min\{\text{msilm}_{A,B}^{i} \text{ for all neighbor interfaces } B \neq A \text{ of router } i\}$$

Based on the minimal return path metrics, we are able to give a criterion which is sufficient to rule out that a path is a Source Loop.

Theorem 1. *(Simple Loop Test)*
Let $P_B^{i,d}$ be the path to subnet d with the lowest metric $m_B^{i,d}$ and $P_A^{i,d}$ another path to subnet d with metric $m_A^{i,d}$, and $B \neq A$. Then the path $P_{A,B}^{i,d,i}$ is a Simple Loop if the following inequality holds:

$$m_A^{i,d} < mrpm_A^i + m_B^{i,d} \tag{1}$$

Proof. The path $P_{A,B}^{i,d,i}$ consists of the constituents $P_A^{i,d}$ and $P_B^{i,d}$ with $m_{A,B}^{i,d,i} = m_A^{i,d} + m_B^{i,d} - 1$. If $P_{A,B}^{i,d,i}$ is a Source Loop, the constituent $P_A^{i,d}$ passes router i and consists of $P_{A,C}^{i,i}$ and $P_D^{i,d}$ for some interfaces $A, C, D \in IF$ with $m_A^{i,d} = m_{A,C}^{i,i} + m_D^{i,d}$. mrpm_A^i is the minimal return path metric via A, so that $m_{A,C}^{i,i} \geq \text{mrpm}_A^i$. $m_D^{i,d} \geq m_B^{i,d}$, because $m_B^{i,d}$ is the lowest metric to subnet d by precondition. This results in $m_A^{i,d} = m_{A,C}^{i,i} + m_D^{i,d} \geq \mathrm{mrpm}_A^i + m_B^{i,d}$. So, if $m_A^{i,d} < \mathrm{mrpm}_A^i + m_B^{i,d}$, the path $P_{A,B}^{i,d,i}$ is not a Source Loop and therefore must be a Simple Loop. □

In order to form a Source Loop, the lower limit of $m_A^{i,d}$ is the sum of the minimal return path metric mrpm_A^i and the metric $m_B^{i,d}$ in order to reach destination subnet d via neighbor interface B. If the inequality holds, the path $P_{A,B}^{i,d,i}$ is simply *too short* to be a Source Loop and therefore must be a Simple Loop. We call the verification that this inequality holds the *Simple Loop Test*. The metric of this path $P_{A,B}^{i,d,i}$ is:

$$m_{A,B}^{i,d,i} = m_A^{i,d} + m_B^{i,d} - 1 \tag{2}$$

If the Simple Loop Test holds for a path $P_{A,B}^{i,d,i}$, we have detected a Simple Loop of metric $m_{A,B}^{i,d,i} = m_A^{i,d} + m_B^{i,d} - 1$.

Figure 1a shows a Simple Loop. There is a path $P_B^{i,d}$ with metric $m_B^{i,d} = 2$ and a path $P_A^{i,d}$ with metric $m_A^{i,d} = 2$, too. The minimal return path metric mrpm_A^i is 3 which yields:

$$m_A^{i,d} = 2 < 3 + 2 = \mathrm{mrpm}_A^i + m_B^{i,d}$$

The inequality is satisfied, the Simple Loop Test is successful and therefore it is proved that $P_{A,B}^{i,d,i}$ is a Simple Loop.

Figure 1b shows a concrete Source Loop of Figure 5c's type. The topology of the network is the same as in Figure 1a, so again mrpm_A^i is 3, but we choose a different subnet $d = s_5$. The path $P_C^{i,d}$ has the metric $m_C^{i,d} = 2$ and the metric of path $P_A^{i,d}$ is 5. In this case the inequality is not fulfilled and the Simple Loop Test fails:

$$m_A^{i,d} = 5 \not< 3 + 2 = \mathrm{mrpm}_A^i + m_B^{i,d}$$

Figure 1c shows a Source Loop in a different network topology. It is of Figure 5a's type. The path $P_A^{i,d}$ consists of the Simple Loop $P_{A,B}^{i,i}$ and the path $P_D^{i,d}$. The Simple Loop Test is always performed using the path with the lowest metric to the destination subnet, which is in this case $P_B^{i,d}$. Given that the metric $m_B^{i,d} \leq m_D^{i,d}$ it is evident that no Source Loop can pass the Simple Loop Test:

$$m_A^{i,d} = 5 \not< 3 + 2 = \mathrm{mrpm}_A^i + m_B^{i,d} \quad (\leq \mathrm{mrpm}_A^i + m_D^{i,d})$$

Again the Simple Loop Test fails.

So far we have analyzed the properties of paths in a static network. We developed the Simple Loop Test in order to identify Simple Loops out of the metric of a closed path in a network. To apply these considerations to distance vector routing, we have to take the distance vector routing process into account. The metrics to destination subnets are sent via update messages between adjacent routers throughout the network. So we have to look at the potential succession patterns of update messages and the chronological sequence in which these messages arrive at a router. However, the Simple Loop Test together with the split horizon rule is sufficient to prevent the acceptance of any update message that produces a Source Loop. To prove this, we examined all possible Source Loop types produced by a simple combinatorial scheme.

	$V = B$	$V = A$	$V = ANY$
$U = A$	ESH	ISH+ESH	ESH (fig.3b)
$U = B$	ISH	SLT	SLT
$U = ANY$	SLT (fig.3c)	SLT	SLT (fig.3a)

Table 1: Four of the nine Source Loop types (see Figure 5) are prohibited by the application of the split horizon rule at the local router (internal split horizon, ISH) or the split horizon rule at the neighbor router (external split horizon, ESH). The remaining five Source Loop types are detected by the application of the Simple Loop Test (SLT).

Assume an update message arrives at router i via neighbor interface A containing the metric $m_A^{i,d}$ to subnet d. If router i has already an entry in its routing table to subnet d via a neighbor interface $B \neq A$ and metric $m_B^{i,d}$, we can build a closed path $P_{A,B}^{i,d,i}$ with metric $m_{A,B}^{i,d,i} = m_A^{i,d} + m_B^{i,d} - 1$. If this path $P_{A,B}^{i,d,i}$ is a Simple Loop, there is a loop in the network between the neighbor interfaces A and B of router i. However, it is also possible that $P_{A,B}^{i,d,i}$ is a Source Loop.

If the path $P_{A,B}^{i,d,i} = (H_1, H_2, \ldots, H_l)$ is a Source Loop, the router i is passed in between via two neighbor interfaces U and V of router i. Therefore, H_n and $H_{n+1}, (1 < n < l,\ I_n, O_{n+1} \in i)$ exists in the sequence of hops with $H_n = (\mathbf{U}, s_n, I_n)$ and $H_{n+1} = (O_{n+1}, s_{n+1}, \mathbf{V})$ (Figure 5).

In this case router i must have received an update message from some neighbor interface V sometime in the past and sent out an update message to some neighbor interface U containing metric $m_V^{i,d}$. This information must have traveled through the network in a loop, finally returning to router i from neighbor interface A. Since U or V may be neighbor interfaces A, B, or any other neighbor interface of router i, there are $3^2 = 9$ possible Source Loop types. Thus, the occurrence of a Source Loop is either prohibited by the split horizon rule or can be detected by the Simple Loop Test (Table 1).

If $U = V$, the split horizon rule of the local router i (internal split horizon) eliminates the possibility that such a Source Loop occurs. If $U = A$, the split horizon rule of the neighbor router with interface A (external split horizon) eliminates the possibility that such a Source Loop occurs (Figure 5b). If $U = ANY$ and $V = B$ (Figure 5c), the Simple Loop Test detects a Source Loop like in Figure 1b. And finally, if $U \neq A$ and $V \neq B$ (Figure 5a), the Simple Loop Test detects a Source Loop like in Figure 1c.

4.3 RMTI

The basic functionality of RMTI is the evaluation of redundant routing information which would usually be rejected immediately by the router. The RMTI processes are independent of the underlaying distance vector protocol. There is no need for an additional or altered communication between routers enhanced with RMTI. RMTI improves the underlaying routing protocol and allows it to maintain or enlarge the existing network infrastructure.

Up to now, we have implemented RMTI on the basis of the Routing Information Protocol (RIP). Thus, our RMTI-protocol is compatible to RIP. RIP works as follows: Assume that a RIP router i receives a routing update from an adjacent RIP router j to subnet d with metric $m^{j,d}$ via interface A of router j. This indicates the existence of a corresponding path $P^{j,d}$ from RIP router j to subnet d. Router i does not know the complete path $P^{j,d}$, but knows the number of hops this path consists of. Then, from the view of router i, there will be a path $P_A^{i,d}$ with metric $m_A^{i,d} = m^{j,d} + 1$ by prepending a hop from i to j to the path $P^{j,d}$. Therefore,

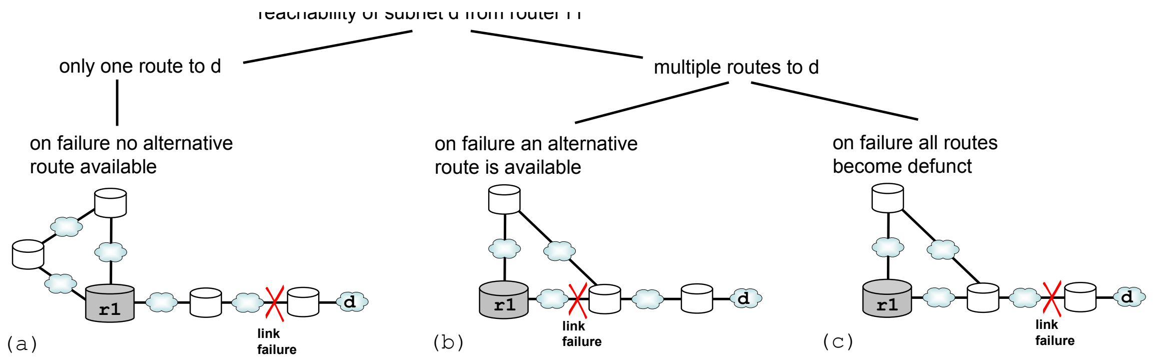

only one route to d
multiple routes to d
on failure no alternative route available
on failure an alternative route is available
on failure all routes become defunct
r1
d
link failure
(a)
(b)
(c)

Figure 6: The basic topologies containing a network loop. The position of the loop and the location of a link failure determines the availability of alternative routes

router i knows the metric $m_A^{i,d}$ and the first hop toward d. If a RIP router has a valid path to subnet d, it will reject all equivalent or inferior paths to the same subnet.

RIP has three timers: an update timer (default 30 sec) for sending out routing updates periodically, a timeout timer (default 180 sec) to recognize invalid entries in the routing table and to mark them as unreachable when the timer expires, and a garbage collection timer (default 120 sec) to completely delete an entry from the routing table.

The drawback of distance vector routing in general, and RIP in particular, is the vulnerability to routing loops and, in correlation to that, the CTI problem. Due to this CTI problem, the RIP specification limits the distance of a route to a subnet to the maximum metric of 15 hops. Metric 16 marks the subnet as unreachable and is defined by the RIP term *infinity*. This restriction does not avoid the CTI problem but reduces its duration and, therefore, its impact on the network. Besides that, it also limits the maximum size and complexity of RIP networks. The split horizon technique only avoids routing loops between two directly connected routers (two-hop loops), but fails within topology loops.

In addition to the usual operation of distance vector routing, RMTI builds up two tables containing information about topology loops in the local network environment. The msilm-table contains upper bounds for the minimal Simple Loop metrics between every two neighbor interfaces and the mrpm-table contains the minimal return path metrics mrpm_A^i for all neighbor interfaces of the local router i.

As shown in Figure 1a, assume that router i has an entry to subnet d via neighbor interface B with metric $m_B^{i,d}$. This indicates a path $P_B^{i,d}$. Now i receives an update to subnet d via neighbor interface $A \neq B$. This indicates an alternative path $P_A^{i,d}$ to subnet d with metric $m_A^{i,d}$. A combination of these paths results in a closed path $P_{A,B}^{i,d,i}$ with metric $m_{A,B}^{i,d,i} = m_A^{i,d} + m_B^{i,d} - 1$. We perform the Simple Loop Test to verify that $P_{A,B}^{i,d,i}$ is a Simple Loop. If this test is passed, we have detected a Simple Loop with metric $m_{A,B}^{i,d,i}$. If this Simple Loop is the first one between the interfaces A and B that we have detected so far, or it has a lower metric than all other detected Simple Loops between A and B, we update the entry in the msilm table and calculate the new minimal return path metrics mrpm_A^i and mrpm_B^i.

Due to the fact that the mrpm entry corresponds to the metric of a minimal Simple Loop and the mrpm entries are needed in the Simple Loop Test, the question of initialization has to be considered. At the beginning, the initial upper bound values for the minimal Simple Loop metrics (msilm) and the minimal return path metrics (mrpm) are set to $2 * \text{infinity} - 1$ (with the usual value infinity$= 16$ this is 31), indicating that no actual upper bounds have been calculated out of updates so far. In addition, we have to perform the Simple Loop Test by replacing the initial value for mrpm_A^i with 2 in order to perform the strictest possible application of this test (2 is the metric of the smallest possible Simple Loop generally). Thus the Simple Loop Test is reduced to test $2 > |m_A^{i,d} - m_B^{i,d}|$ in this case. Every Simple Loop contains at least one subnet which is located opposite to the considered router in the topology loop. Both neighbor routers, which span the Simple Loop over the underlaying topology loop, advertise a route to this subnet and the difference between the metrics of the two routes will be smaller than 2. So there is always a subnet d, that allows the detection of a Simple Loop in the initial case. Once the initial Simple Loop Test is passed successfully, the metric of the first Simple Loop can be calculated on the metrics of the two corresponding routes. By the time the routing process is converged, every router has detected all local Simple Loops. This loop knowledge now enables RMTI to effectively meet all challenges during the convergence phase of a network after link failures and topology changes, as the next section will show.

5. PERFORMANCE AND PARADIGMS

Topology loops provide connection redundancy in order to ensure reliability and stability of the network. Figure 6 shows basic topologies containing a topology loop. Three basic cases are illustrated there. If a network topology contains loops, certain subnets are reachable from a router by more than one link. In case of a link failure on the preferred route, there might be an alternative route to the subnet.

The answer to the question, whether or not there is an alternative route available from router r_1 to subnet d after a link failure, depends on the position of the loop and the location of the link failure in the topology.

In Figure 6a, there exists only one route to subnet d from router r_1. In case of a link failure to that route, there is no alternative route available to subnet d despite the existence of the loop. In the topology of Figure 6b, the loop provides two possible routes from router r_1 to subnet d. There is an alternative route available if a link failure occurs within this loop. However, if the link failure occurs beyond this loop, all routes to subnet d from router r_1 become defunct and no alternative route exists (Figure 6c).

If router r_1 is a conventional RIP router, the topology of Figure 6a is prone to routing loops and the CTI problem. Router r_1 accepts malicious routing updates to subnet d from its neighbors. Section 5.3 presents the test results with RMTI instead of RIP in operation and demonstrates the main advantage of RMTI over RIP. Complex network topologies with multiple loops are composed of the basic topologies in Figure 6. Section 5.4 demonstrates the ability of RMTI to handle these complex network topologies. The test results show that RMTI is far superior in solving routing decisions as compared to standard RIP.

We implemented RMTI on top of an existing RIP router in order to perform the following performance tests. Thus, we are able to demonstrate the applicability of the RMTI approach in real deployment environments and substantiate its benefits, especially compared to standard RIP.

We use the well-known and widely used Quagga routing software suite to enable and explore the routing process. The Quagga routing suite consists of several advanced routing software daemons for Linux and Unix-like systems, where the *ripd* daemon implements the RIPv2 protocol used as the underlaying routing protocol. We additionally implemented a communication link in a network test environment which allow us to influence the succession of routing updates and evaluate network situations frequently.

5.1 The Test Environment

Figure 7 shows the operation schema of our test environment we developed in order to evaluate the performance of our RMTI daemon. We extended our routing daemon by a client-server communication to a supervisor which allows us to influence the succession of routing updates and the logging of all local information which appear on the routing daemon. Thus, we get a comprehensive representation of the available network information. The supervisor allows us to manipulate the routing behavior of every routing daemon by triggering updates in a predefined sequence in order to test critical network situations like the CTI problem. Our evaluation focused on topologies containing a loop. In

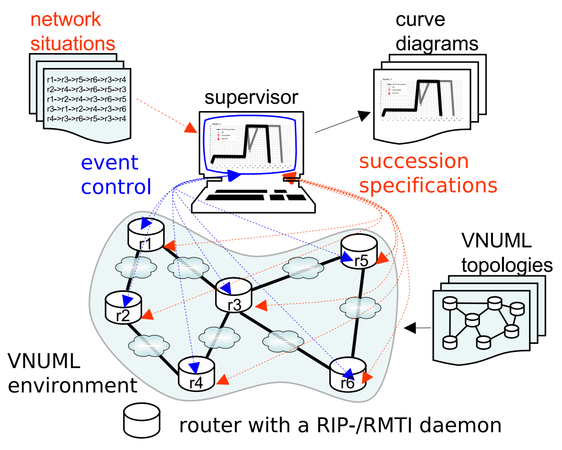


Figure 7: The operation scheme of our test environment

topologies without a loop RMTI works just as well as standard RIP. We examined both routing daemons on a huge number of various network scenarios to verify the benefit of RMTI when it was exposed to the CTI problem and to verify the equality in the absence of the CTI problem. We performed a frequent modification of the network topology by using the open source virtualization tool *Virtual Network User Mode Linux* (VNUML) [19] which is based on *User Mode Linux* (UML) [4]. Using User Mode Linux is an efficient way to run various Linux systems inside one physical machine and connect them via virtual interfaces to build up arbitrary network topologies. VNUML is a comfortable tool to quickly design and boot up virtual network scenarios. A VNUML network can be administered and used just like a network with real Linux machines. Virtualization techniques are especially good for setting up arbitrary network topologies. They provide an opportunity to utilize real operating systems and routing software without the need for all the hardware like routers, switches and the cabling. Apart from this, a modification of the virtual network topology is much easier and quicker to perform than with real networks. Virtualization techniques are particularly suited to support the study of routing performance in complex network topologies. As we didn't want our RMTI daemon to become heavily optimized for virtual networks, we additionally connected real router hardware to our VNUML virtual networks. We used Linksys WRT54G routers equipped with the open source router operating system *openwrt* [20], which is a Linux-based routing system. Thus, we also tested the functionality of our RMTI daemon on real router hardware. VNUML additionally offers a complex but flexible network topology.

Our supervisor consists of a routing update generator which can coordinate all updates, so that all crucial succession patterns (latency and timing issues) in a specific topology will be exhausted. First, a topology of routing daemons has to be chosen and launched by VNUML. Second, all paths which contain a loop have then to be identified and represented. Third, this knowledge is used by the generator to control the updates of every routing daemon, to provoke a situation where the CTI problem will arise. Additionally all events are logged. After these initial conditions have been established, all routers are switched back to automatic mode, where they are no longer controlled and act autonomously. The provocation of a CTI causes the one update with old information to be injected into the topology loop, and the routing protocol has to react to this event. As shown in this section, RIP propagates faulty information and the course of CTI events begins within the topology loop. Metric changes are displayed directly in the test environment so CTIs can be observed or detected automatically. The curve diagrams depicted in this section describe the metric progression of a considered route on a distinct router in different network scenarios. All network scenarios described in this section were performed in our test environment and analyzed with our supervisor.

5.2 The Counting to Infinity Problem

In the following we demonstrate the occurrence of the CTI problem within the network scenario exemplified by Figure 8, which is the basic topology of Figure 6a. Starting from router r_1, which holds a route to subnet d via neighbor interface B with metric $m_B = 3$, we assume that the link to

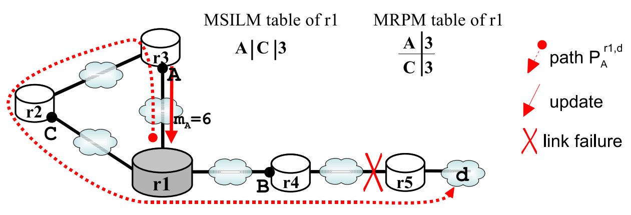


Figure 8: The CTI problem arises.

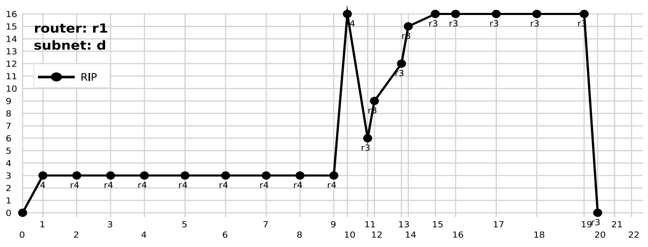


Figure 9: The curve follows the metric of the route to subnet d on router r_1. It depicts the course of CTI events.

subnet d via router r_4 becomes corrupted. This is announced by an update which router r_1 receives from r_4 and, therefore, the corresponding route in r_1's routing table is marked as unreachable. Next r_1 receives a routing update via neighbor interface A from router r_3 advertising an alternative route to subnet d (via r_2) with metric $m_A = 6$. However, the routing information of this update was propagated along a Source Loop.

If router r_1 is a conventional RIP router, it will accept this faulty routing update from r_3, adapt its routing table, and propagate the new routing information. Then a routing loop is established and the course of CTI events will begin. Moreover, as long as the CTI events have not come to an end, a forwarding loop exists and a large amount of network bandwidth is consumed, due to data packets having been sent to subnet d and circulating in the forwarding loop. The CTI events also suppress the propagation of the correct routing update announcing the subnet as unreachable. Existing RIP extensions like triggered updates would speed up the CTI problem but in the worst case routing updates would get lost and the elimination of the routing loop would require more time. The RIP extension Split Horizon also does not avoid the CTI problem in this situation.

The curve chart in Figure 9 refers to the network situation described in Figure 8. It shows the metric values of the route to subnet d, captured on the router r_1 while the events of the CTI sequence are progressing. The y-axis represents the metric of the route to subnet d in the range from 0, as the destination subnet is unknown to the router r_1, to 16, as the destination subnet is known with metric 16 (infinity). The x-axis represents the number of check and update events in relation to the corresponding route entry in the routing table of the regarded router, e.g., incoming updates. Metric changes are directly displayed in a time-synchronous manner. The curve chart is generated automatically by our supervisor.

The curve chart in figure 9 shows the metric of the route to subnet d on router r_1 and describes the CTI progression.

As depicted in the chart, router r_1 holds a route to subnet d via router r_4 with metric 3 (x-axis 1-9). Just before the CTI arises, the route to subnet d becomes corrupted (x-axis 10) and router r_1 marks it with infinity (metric 16). Next, router r_1 gets a supposed alternative route to subnet d from router r_3 with metric 6 (x-axis 11). However, this route is outdated and invalid and belongs to a path which already includes router r_1. If router r_1 is equipped with standard RIP, it accepts the route to subnet d via router r_3 and a routing loop occurs. The sequential incrementation of the metric up to infinity (x-axis 12-15) is the characteristic of the CTI problem.

5.3 Basic Topologies

The CTI problem in Figure 8 is prevented by router r_1, if it is enhanced with our RMTI approach. Regarding the situation in Figure 8, the routing update from r_3 will be rejected by r_1 and subnet d is kept unreachable in r_1's routing table because there is no Simple Loop spanned over router r_3 and r_4.

The test results captured on router r_1 are shown in Figure 10. The separate, unfilled, downward pointing triangle indicates that RMTI on r_1 receives and rejects the *critical update* that would have induced the CTI problem (gray curve in the background of Figure 10). The symbols and curve lines used are explained in the corresponding legend of each curve chart.

In the network situation of Figure 8, the RMTI router r_1 decides whether the update from router r_3 should be accepted or not after a link failure. As there is no Simple Loop between r_3 and r_4, the same information cannot be advertised from both routers.

The following case study (see Figure 11) refers to a pitfall of Simple Loop detection. It is not sufficient to perform the Simple Loop Test (Equation 1, Section 4.2) just with the last valid entry for a subnet d in the forwarding table of a router r_i. This may lead to detecting a Source Loop as a Simple Loop by mistake. Instead, we always have to use the smallest metric for subnet d recently known by r_i.

Starting from router r_1, which holds a route to subnet d via r_2, we assume that subnet d becomes unreachable due to a link failure behind r_2. Router r_2 sends a routing update with metric 16 to router r_1 and r_4. Router r_1 adapts its routing table and advertises the new information about subnet d to its neighbors. Before router r_4 receives this routing update from r_2, it sends a routing update with old invalid reachability information about subnet d to r_1. Router r_1 accepts the routing update from r_4 as seemingly new information about subnet d.

Additionally, we assume the situation is similar to the CTI

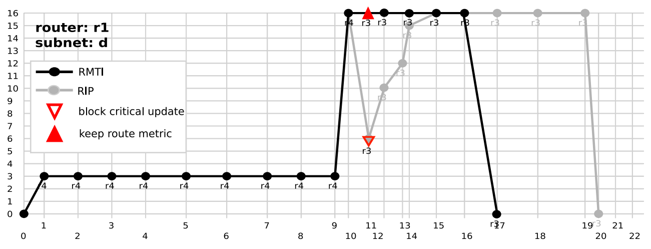
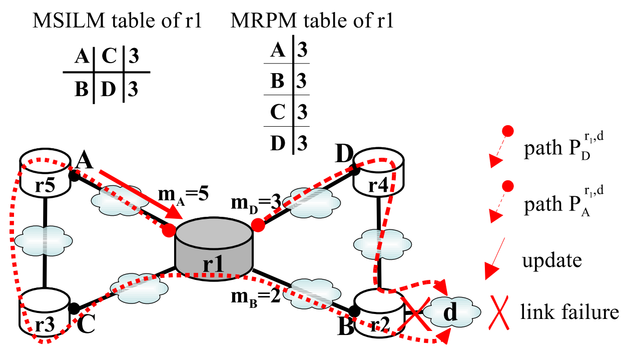


Figure 10: The CTI is prevented by RMTI router r_1 discarding the faulty route information from r_3.



MSILM table of r1

A	C	3
B	D	3

MRPM table of r1

A	3
B	3
C	3
D	3

Figure 11: The CTI problem arises. In this situation using the right values to perform the Simple Loop Test is essential to detect all Simple Loops correctly.

problem in Figure 8. Router r_5 receives the routing update with metric 16 which was sent from r_1 before. Router r_5 adapts its routing table and marks the route to subnet d as unreachable. However, router r_3 does not receive the routing update from r_1 and keeps its route to subnet d as reachable. Next, router r_5 receives a routing update from r_3 with seemingly alternative reachability information about subnet d. Router r_5 accepts this route information and advertises it to r_1. Although router r_1 does not accept the routing update, it detects a Simple Loop between its neighbor interfaces A and D. This is because r_1 receives information about subnet d from r_4 with metric $m_D = 3$ and from r_5 with metric $m_A = 5$ and the Simple Loop Test of r_1 is passed with $m_A < mrpm_A + m_D \Rightarrow 5 < 3+3$. Hence, a CTI could occur, due to a falsely detected and stored Simple Loop.

We handle such situations by not immediately adjusting the internal data of RMTI with every accepted routing up-

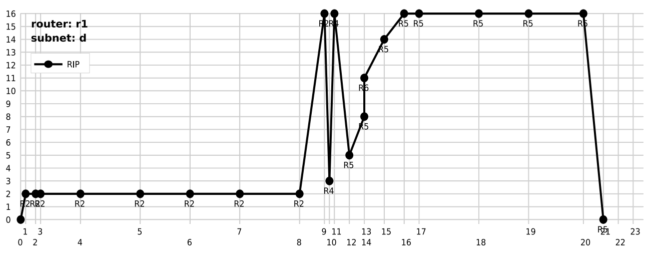


Figure 12: The curve diagram of the CTI problem in relation to Figure 11.

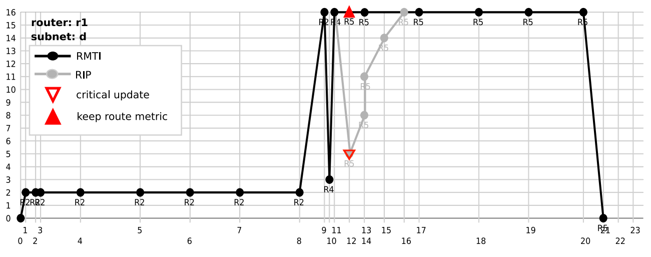


Figure 13: The CTI problem is prevented by RMTI using always the shortest route to the destination subnet of the recent past.

date that changes the routing table. The RMTI has to be performed with the smallest metric of the route to subnet d seen in a short time interval before the metric is changed to a higher value or is set to infinity. Therefore, we have to store the smallest metric of the recent past. The time interval should depend on the RIP update timer interval. In this case r_1 receives redundant information about subnet d from r_4 and r_5 but RMTI still uses the information about subnet d from r_2 with metric $m_B = 2$. Thus, the Simple Loop Test fails with $m_A < mrpm_A + m_B \Rightarrow 5 < 3+2$ and the Source Loop is detected.

If router r_1 marks its route to subnet d as unreachable and gets a routing update from r_5 with seemingly new information about subnet d, r_1 recognizes the malicious update by the missing Simple Loop between its neighbor interfaces A and B. The curve chart in Figure 12 shows the succession of metric changes while the CTI problem occurs. However, as shown in Figure 13, although the routing information which returns to the router is no longer available in the routing table, RMTI can prevent the CTI problem.

5.4 Complex Topologies

In complex topologies the Simple Loop Test is needed (see Equation 1, Section 4.2) to prevent the CTI problem. In Figure 14, the network topology that was shown in Figure 8 is extended by a link between router r_3 and r_4. In this topology, a routing update sent from router r_3 to r_1 might contain valid reachability information about subnet d because of the existing Simple Loop. If r_1 loses the route to subnet d via r_4, subnet d could be either still reachable or completely unreachable depending on the unknown location of the failed link. If the link between r_1 and r_4 fails, subnet d will still be reachable for r_1 via r_3. Whereas if the link between r_4 and r_5 fails, subnet d will become unreachable for r_1. However, if the CTI problem occurs, RMTI can still cope with these situations.

If the link between router r_4 and r_5 is corrupted and subnet d is not reachable anymore, then router r_4 will propagate a routing update to its neighbors r_1 and r_3. Both accept the new route information from r_4 and mark their route to subnet d as unreachable. But router r_2, having learned the route from r_1, sends a routing update with old route information to r_3. Router r_3 accepts this update from r_2 as a new valid route and replaces the old route entry in its routing table. If the routing update is then announced back to r_1, a Source Loop will appear. The RMTI router r_1 detects this Source Loop and prevents the CTI problem by rejecting the routing update from router r_3. The Simple Loop Test of r_1 fails with $m_A < mrpm_A + m_B \Rightarrow 6 < 3+3$. Therefore, the Source Loop is detected due to the inappropriate metrics.

However, due to an invalid old routing update from r_3 very shortly after the link failure was perceived by r_1 but not yet by r_3, router r_1 accepts the old invalid information about subnet d from r_3 as a seemingly valid alternative route. This cannot be prevented because the routing update from r_3 could be invalid old route information as well as a valid alternative route information, e.g., if either the link between r_4 and r_5 or the link between r_1 and r_4 became corrupted.

Again, the major question in this situation concerns the data used in the Simple Loop Test. Assuming router r_1 and r_3 lost their route to subnet d. Then shortly after r_1 marks the route as unreachable, it accepts a faulty old routing update from r_3 with invalid route information. Router

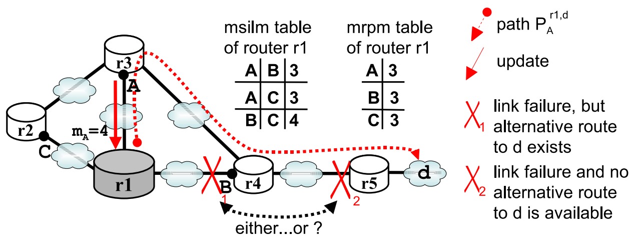

msilm table of router r1
mrpm table of router r1
path $P_A^{r1,d}$
update
link failure, but alternative route to d exists
link failure and no alternative route to d is available
either...or ?
m_A=4

Figure 14: Such a faulty routing update is generally difficult to detect for routers because the location of the link failure creates an ambiguous situation. However, the acceptance of the update does not induce a CTI in this case and does not confuse RMTI router r_1 which can still prevent the occurrence of the CTI problem.

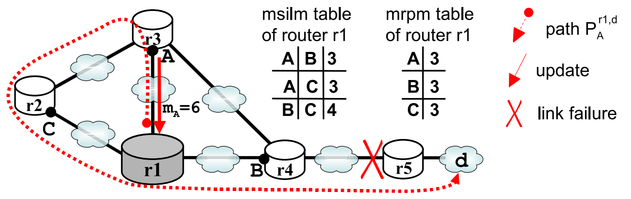


Figure 15: RMTI router r_1 is able to detect this routing update via the Simple Loop Test.

r_3 immediately corrects this by sending another routing update announcing subnet d as unreachable. Router r_1's last valid route to subnet d describes a path via r_3 whereas r_2 still has the old route information from r_1. If this route information is announced back to r_1 in a Source Loop via r_3, as described in Figure 15, the Simple Loop Test in r_1 performs $m_A < mrpm_A + m_B \Rightarrow 6 < 4 + 3$ and is passed with $6 < 7$. Hence, if RMTI router r_1 would perform the Simple Loop Test in this way, simply with the latest valid metric of the route from the routing table, it would accept the routing update and the CTI problem would occur in this situation.

With the topology explained in Figure 11, we have shown how to handle such situations by not immediately adapting new route information from the routing table into RMTI. The behavior of the Simple Loop Test is stricter with lower metrics than with higher metrics. In this situation RMTI ignores the routing update from r_3 with metric 4 long enough and prevents the CTI problem. The metric progression of this network situation is shown in Figure 16.

5.5 Indistinguishable Routes

A special case is shown in Figure 17 where the decision to accept or to reject a route not only depends on the length of the routes' metrics. It may be possible that the metric of an alternative route is equal to or even larger than the limit given by the Simple Loop Test. As shown in Figure 17, the routing updates may contain a valid alternative route to subnet d (dashed line) or an invalid Source Loop (dotted line). Both variants of routing updates advertise the same metric 6 to r_1. The Simple Loop Test of r_1 performs $m_A < mrpm_A + m_B \Rightarrow 6 < 3 + 3$ and fails with $6 < 6$. Therefore, the routing update is rejected by the RMTI router r_1.

Figure 17 shows a Source Loop (dotted line) and an alternative path (dashed line). Router r_1 cannot distinguish the difference between these paths due to the same metric. In order to solve this decision problem, RMTI uses a timer to limit the execution time of the Simple Loop Test. RMTI supposes that Source Loops can be deleted by sending a routing update with metric infinity. Then a timer is started for a given time interval and further incoming routing updates with a valid metric will be blocked. If within this time interval a routing update with metric 16 is transmitted by r_3, a Source Loop has just been deleted and a CTI is prevented. If the timer expires and the route is still offered with a valid metric, it has to be a valid route. As the critical period for CTI situations is rather short, the timeout interval

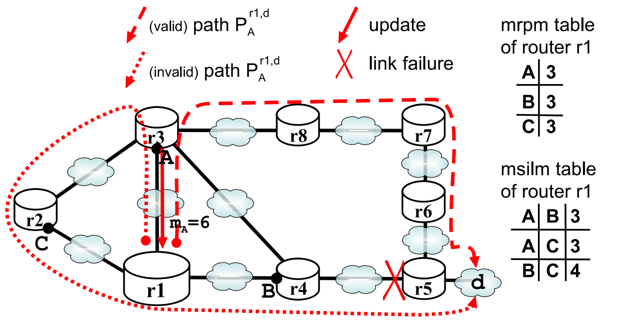


Figure 17: RMTI has to distinguish between a valid and an invalid routing update with the same metric announced via the same neighbor interface.

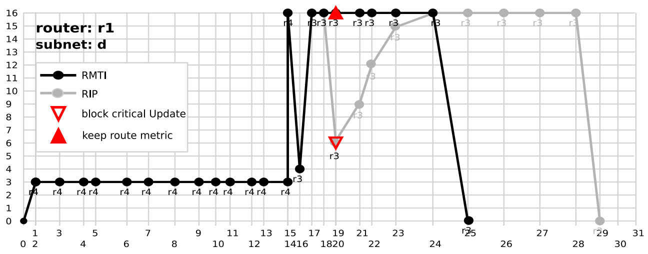


Figure 16: The RMTI ignores routing updates which are advertised directly after a route failure because they could cause confusion. The routing update from r_3 (x-axis 16) is ignored by RMTI, otherwise the CTI problem could not be prevented.

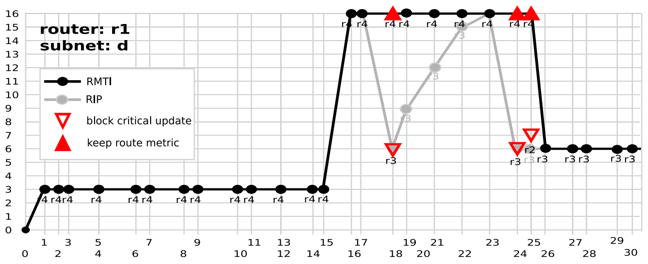


Figure 18: RMTI mainly decides on metrics. However, if a route is refused by RMTI, it will be blocked just for a short time interval. During this time interval, an existing Source Loop should be deleted by a transmitted routing update.

Time	Route Status	RIP	RMTI
Start	Insert the route into the routing table and use it for forwarding.		
Phase 1	Default timeout timer is reinitialized with every incoming update. By default the timeout timer expires after 6 missing updates. (route metric $<$ infinity)	Shorter routes overwrite longer routes.	Shorter routes overwrite longer routes. The minimal Simple Loops are identified and stored
The route becomes invalid (e.g., timeout timer expires)	Set the metric to infinity and do not use it for forwarding. Stop the timeout timer and start the garbage collection timer.		
Phase 2	Default garbage collection timer will be stopped by an incoming update with a metric smaller than infinity. (route metric = infinity)	The first routing update advertised with a metric smaller than infinity is accepted as alternative route.	Only routes in a Simple Loop with the route's adjacent subnet are accepted as alternative routes.
Garbage Collection Timer expired.	Delete the route from the routing table.		

Table 2: Comparison between RIP and RMTI operation phases

of the timer can be short, too. The timer could depend on the routing update interval or on the metric of the Simple Loops involved.

The curve diagram in Figure 18 shows the test results of the RMTI router within the network situation described in Figure 17. First, the CTI problem is avoided and no alternative route to the destination subnet (x-axis 18-23) exists. Second, a valid alternative route is advertised by router r_3; again r_1 rejects the update for the first time (also an incoming update from router r_2) but then the timer expires, a request is sent, and the alternative route is accepted next (x-axis 24-26).

6. IMPLEMENTATION OF RMTI

Up to now we have implemented RMTI as a slight extension to standard RIP. RMTI simply enhances RIP on detection of topology loops and prevention of routing loops. There is no need to change the RIP interaction behavior or any of the RIP message formats. Therefore, RMTI is compatible with RIP and both can be deployed on routers in the same network. In this section we introduce the implementation of RMTI in more detail and describe the changes which must be made in an existing RIP implementation. The template of a RMTI implementation which is described here is based on our existing RMTI daemon implementation. RMTI is implemented as an extension to the RIP update procedure of the routing table in which the present route information is compared with incoming new route information about the same destination subnet.

The control flow of the entire RMTI process is described in a flow chart in Figure 19. As shown in Figure 19a, the RIP implementation has to be extended at a particular point where the RMTI enhancements are then implemented. The control flow diagram is divided into a data collection phase (Figure 19b) and a decision phase (Figure 19c) in relation to the two distinct phases RMTI mainly operates in. Simple Loops will be detected in the data collection phase whereas Source Loops will be prevented in the decision phase. If the present route in the routing table is valid and an alternative route to the same subnet is advertised to the RMTI router, the metric of the corresponding path composed of the two routes will be calculated by equation 2 (see Section 4.2). The obtained path is a Simple Loop if equation 1 holds. Only the Simple Loop with the smallest metric is stored as the minimal Simple Loop of the corresponding two neighbor interfaces for further use. This happens as long as the route to the destination subnet is still available with a valid metric. When the route becomes invalid and is marked as unreachable in the routing table, the information about the existing Simple Loops is used to decide, whether or not to accept an advertised alternative route to the subnet via another neighbor router. Table 2 summarizes all operation phases and compares the behavior of RMTI with standard RIP.

constant name	meaning
RMTIinfinity	RIP infinity $*2$
RMTIalive	RIP timeout timer + RIP garbage collection timer

Table 3: RMTI constants

variable name	meaning
destination	the IP-address of the destination subnet of the route.
metric (m_B)	the distance to the destination subnet.
nexthop	the next hop router (or gateway) along the route to the destination, this is the IP-address of the neighbor interface.
timer	the amount of time since the entry was last updated.
rmti_oldmetric	the last valid metric, used to calculate the metric of the path combination when the route is timed out and the metric is set to infinity.
rmti_oldnexthop	the corresponding next hop to the rmti_oldmetric variable.
rmti_validity	a boolean variable which indicates the phases the RMTI protocol operates in (RMTI data collection phase or RMTI decision phase).
rmti_synctime	a time stamp variable which specifies a time interval to control the point when the old metric and the old next hop should be adjusted with the information in the routing table.
rmti_checktime	a time stamp variable which marks the route in the routing table to recognize route advertisements which have already been rejected before.

Table 4: The variables in the route structure

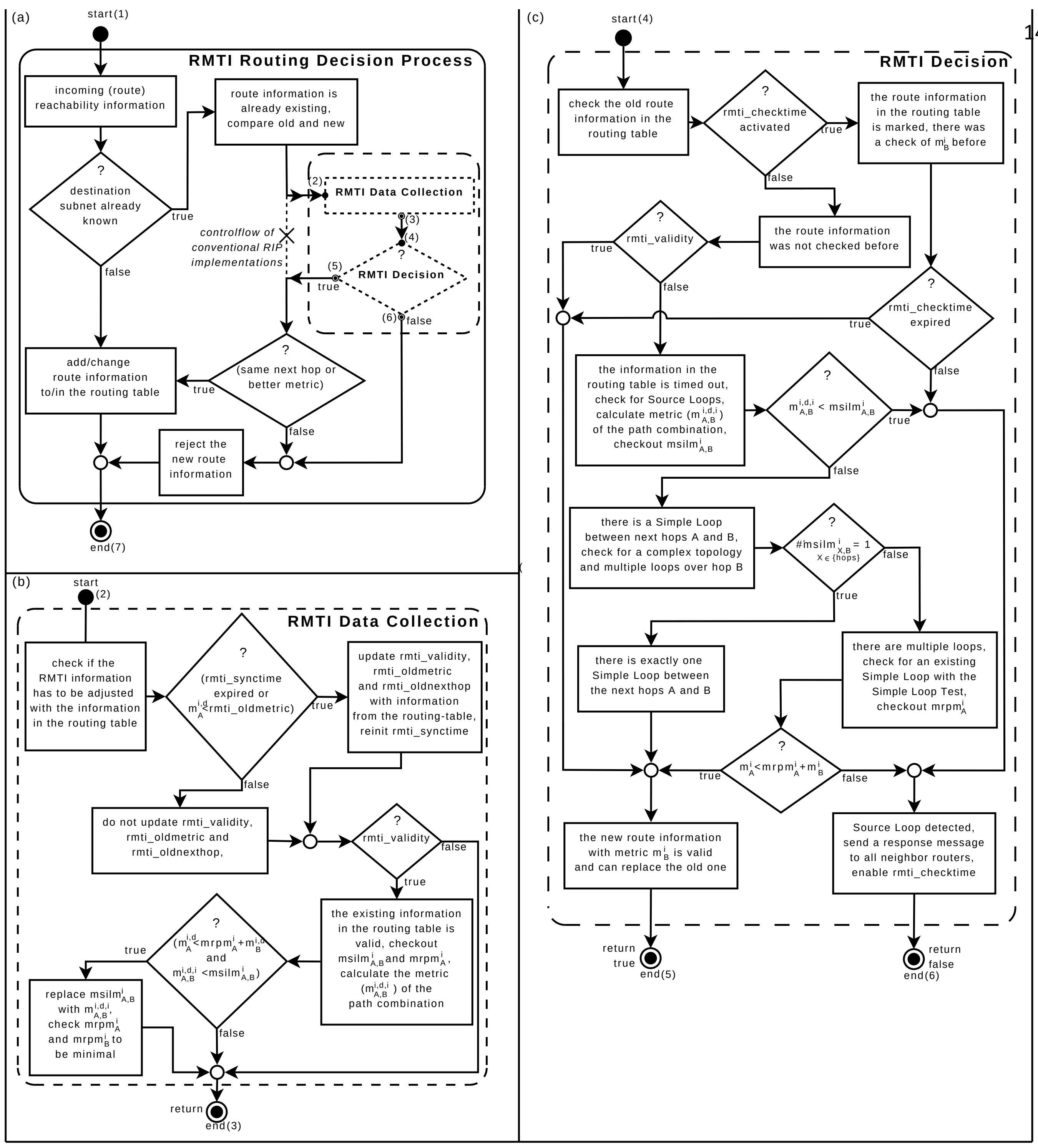

(a)
start(1)
RMTI Routing Decision Process
incoming (route) reachability information
route information is already existing, compare old and new
? destination subnet already known
true
false
(2) RMTI Data Collection
(3)
(4)
? RMTI Decision
(5)
(6)
controlflow of conventional RIP implementations
true
false
add/change route information to/in the routing table
? (same next hop or better metric)
true
false
reject the new route information
end(7)
(b)
start (2)
RMTI Data Collection
check if the RMTI information has to be adjusted with the information in the routing table
? (rmti_synctime expired or $m^{i,d}_A$<rmti_oldmetric)
true
false
update rmti_validity, rmti_oldmetric and rmti_oldnexthop with information from the routing-table, reinit rmti_synctime
do not update rmti_validity, rmti_oldmetric and rmti_oldnexthop,
? rmti_validity
false
true
the existing information in the routing table is valid, checkout $msilm^i_{A,B}$ and $mrpm^i_A$, calculate the metric ($m^{i,d,i}_{A,B}$) of the path combination
? ($m^{i,d}_A$<$mrpm^i_A$+$m^{i,d}_B$ and $m^{i,d,i}_{A,B}$ <$msilm^i_{A,B}$)
true
false
replace $msilm^i_{A,B}$ with $m^{i,d,i}_{A,B}$, check $mrpm^i_A$ and $mrpm^i_B$ to be minimal
return
end(3)
(c)
start(4)
RMTI Decision
check the old route information in the routing table
? rmti_checktime activated
true
false
the route information in the routing table is marked, there was a check of m^i_B before
the route information was not checked before
? rmti_validity
true
false
? rmti_checktime expired
true
false
the information in the routing table is timed out, check for Source Loops, calculate metric ($m^{i,d,i}_{A,B}$) of the path combination, checkout $msilm^i_{A,B}$
? $m^{i,d,i}_{A,B}$ < $msilm^i_{A,B}$
true
false
there is a Simple Loop between next hops A and B, check for a complex topology and multiple loops over hop B
? #$msilm^i_{X,B}$ = 1 X ∈ {hops}
false
true
there is exactly one Simple Loop between the next hops A and B
there are multiple loops, check for an existing Simple Loop with the Simple Loop Test, checkout $mrpm^i_A$
? m^i_A<$mrpm^i_A$+m^i_B
true
false
the new route information with metric m^i_B is valid and can replace the old one
Source Loop detected, send a response message to all neighbor routers, enable rmti_checktime
return true
end(5)
return false
end(6)

Figure 19: Flow charts of the RMTI implementation
(a) The flow chart of the entire RMTI process embedded in standard RIP. The RMTI algorithm can be implemented as a separate procedure independent of the existing implementation.
(b) The procedure of collecting information data by RMTI. If the route to the destination subnet in the routing table is available, the RMTI detects Simple Loops and stores the minimal one.
(c) This is the core procedure of RMTI which is entered only if the route in the routing table is marked with metric 16 (infinity) and might be replaced by the new alternative route information. Depending on the relation between the metric of the combined path of the two routes and the corresponding *msilm* value the return value of RMTI is true if the new route information meets all requirements of RMTI or false if not (see Equation 1, Section 4.2). Nevertheless, if the new alternative route information is accepted by RMTI, it also has to pass the remaining RIP decision procedures to be eventually inserted in the routing table. However, it will be completely discarded if the new alternative route is refused by the RMTI.

6.1 Constants and Data Structures

Based on a RIP implementation as underlaying router platform, there are two new constants necessary for an implementation of RMTI. They are constructed on well-known RIP constants. Table 3 describes the RMTI constants and their relation to the existing RIP constants. RMTIinfinity correlates to the infinity metric of RIP and marks a detected Simple Loop as no longer valid. RMTIinfinity corresponds to the longest possible Simple Loop metric calculated by equation 2 (see Section 4.2), when both routes to the destination subnet are marked as unreachable with metric infinity. As the metric of infinity is 16 in standard RIP, the metric of RMTIinfinity is 31.

The timeout constant RMTIalive corresponds to the number of seconds before a detected Simple Loop is marked as invalid with metric RMTIinfinity. RMTIalive is defined by the sum of the timeout timer and the garbage collection timer of the underlaying standard RIP implementation.

In comparison with RIP, RMTI needs a slightly modified routing table with a few additional constants. The routing table of standard RIP contains routes consisting at least of the destination subnet, the next hop router, and the metric of the route. Furthermore, a timer variable is needed to handle invalid routes. Table 4 describes all variables of a route entry in the routing table, which have to be implemented within RMTI based on RIP. Furthermore, RMTI uses two global tables, called *msilm* and *mrpm* table. Both tables store the metrics of the detected minimal Simple Loops to be used for the decision whether to accept or reject an alternative route advertisement.

The msilm table (see Table 5) stores the *minimal simple loop metric*. An entry in the msilm table consists of the two related neighbor interfaces which are identified by the source IP addresses of the incoming routing updates and the calculated metric of the corresponding Simple Loop. Additionally, a timer variable is used to detect if a Simple Loop is corrupted and the corresponding msilm entry is not valid anymore. Although the complexity of the msilm table increases quadratically with each stored metric, the msilm table remains rather small because the maximum number of Simple Loops depends on the number of neighbors. The msilm table is symmetric, i.e., the table entry $\text{msilm}_{A,B}$ is equal to entry $\text{msilm}_{B,A}$. There is no need to store both values, so the table can be linearized to save storage space and speed up access rates.

The mrpm table (see Table 6) contains the *minimal return path metric* addressed with the one corresponding neighbor interface. The mrpm entry corresponds to the smallest metric of any Simple Loop which starts at the specific neighbor interface and returns at the router via another arbitrary neighbor interface. Thus the mrpm entry is the smallest msilm value of a distinct neighbor interface and could be implemented as a reference to the corresponding msilm entry.

If the topology changes or a neighbor router becomes corrupted, the corresponding mrpm entry must be deleted. Therefore, the validity of the mrpm entry is proved with a timer variable and marked with RMTIinfinity when the timer expires.

6.2 The RMTI Data Collection Phase

As long as the route in the routing table is valid with a metric lower than infinity, RMTI gathers information about Simple Loops in relation to the subnet under consideration and updates the msilm and mrpm tables (Figure 19b). When the RMTI data collection phase is provided with further routing updates, the *rmti_synctime* variable is being evaluated first. The rmti_synctime variable is needed to cope with the problem of malicious updates (see Section 5.3). The information in the msilm and mrpm tables are not adjusted as soon as new routing information is entered in the routing table, but also the rmti_synctime variable has to be expired. If the given time interval is exceeded, it will be reset and the present information in RMTI will be adjusted with the information in the routing table.

Due to malicious routing updates which would affect RMTI negatively, we do not use the very last route information from the routing table but instead the last route information with the smallest metric in the given time interval. If the rmti_validity variable is false (resp. $metric = infinity$), the process will be aborted, because the considered route in the routing table is marked as unreachable and could not be part of a Simple Loop. If the rmti_validity variable is true ($metric < infinity$), RMTI will detect and update Simple Loops.

The Simple Loop Test is used to prove if the new incoming alternative route and the present route in the routing table can be combined to a valid Simple Loop. If the Simple Loop Test passes successfully, the metric of the detected Simple Loop will be calculated.

The Simple Loop with the smallest metric is stored as the minimal Simple Loop metric according to the two corresponding neighbor interfaces in the msilm table. If a msilm entry is changed, the mrpm entries of each neighbor interface must be recalculated. Then the RMTI data collection phase has come to an end and the decision phase has started.

6.3 The RMTI Decision Phase

Figure 19c describes the control flow of the RMTI decision phase. In this phase an alternative route would be completely rejected or passed along to the standard RIP decision process. At the beginning of the RMTI decision phase the rmti_checktime variable is checked in order to handle indistinguishable routes we have explained in Section 5.5. When an identical routing update was rejected by RMTI in the recent past, the rmti_checktime variable contains the time since the first rejection.

variable name	meaning
interfaceA	the IP address of the neighbor interface A
interfaceB	the IP address of the neighbor interface B
$\text{msilm}_{A,B}$	the minimal Simple Loop metric between the neighbor interfaces A and B
timer	the amount of time since the msilm entry was last acknowledged by routing updates

Table 5: The structure of the msilm entry

variable name	meaning
interfaceA	the IP address of the neighbor interface A
mrpm_{A}	the minimal return path metric of neighbor interface A
timer	the amount of time since the mrpm entry was last acknowledged

Table 6: The structure of the mrpm entry

If the value of the rmti_checktime variable exceeds a predefined time span an incoming alternative route will be passed along to the conventional RIP decision process without any further checks by RMTI and the rmti_checktime variable will be set to zero. If the rmti_checktime variable is zero an incoming alternative route will always be checked by RMTI.

If rmti_validity is true (resp. $metric < infinity$), a valid route exists in the routing table and the RMTI decision phase can be aborted without further consideration. If it is not true ($metric = infinity$), the new route information has to be processed to decide if it can replace the existing route information in the routing table. RMTI calculates the metric of the path combination and compares it to the msilm entry $\text{msilm}^i_{A,B}$. If $\text{msilm}^i_{A,B}$ is higher, then no Simple Loop has been detected between the two neighbor interfaces involved and the new route information can be completely rejected. If any Simple Loop exists between two neighbor interfaces, the metric of the smallest Simple Loop is stored in the msilm table. Otherwise, if no Simple Loop exists, the msilm table contains the initial value RMTIinfinity for this pair of neighbor interfaces. If a Simple Loop is stored in the msilm table, the presence of multiple loops must be excluded before the alternative route information can be approved. Dealing with multiple loops is more complex. We have explained the characteristics of complex topologies in Section 5.4.

Complex topologies can be discovered by counting the entries of a neighbor interface in the msilm table. If the neighbor interface of the old existing information has exact one entry listed in the msilm table, it has only one Simple Loop with another neighbor interface. Then the new alternative route information is passed along. If a complex topology is discovered, the Simple Loop Test (Equation 1, Section 4.2) has to be performed. If the Simple Loop Test fails, the new route information must be rejected and the present route information with metric infinity will be kept. If the Simple Loop Test passes, the new alternative route information will be accepted and passed along to the routing decision process of RIP.

As described in Section 5.5, due to indistinguishable routes with equal metrics, a new alternative route information is blocked by RMTI within a short period of time. When the first rejection of a route occurs, the rmti_checktime variable for the existing route in the routing table is activated and a corresponding routing update with metric infinity is sent out to all neighbors. This routing update will delete an existing Source Loop by purging the malicious route entries in all routers involved. Then any correlated CTI problems and routing loops are prevented.

If the present route in the routing table is marked as unreachable and there is a valid alternative route to the same destination subnet with an indistinguishable metric, then the first incoming corresponding routing update would be rejected by RMTI. But the convergence time would not be appreciably impaired due to the rmti_checktime variable.

7. CONCLUSION

It has been shown that our new RMTI protocol can recognize loops and therefore offers better convergence properties than other distance vector routing protocols. On the other hand, it is fully compatible with RIP by evaluating the common RIP updates more carefully. Every router can thus recognize all loops starting and ending at the router. This loop knowledge is used together with the states of the routing table before and after a network change in order to decide on the acceptance or rejection of an incoming new routing update. It has been shown that our RMTI protocol can handle simple and complex topologies as well. Problems like counting to infinity can no longer appear because we can detect every routing update from a router via a loop back to the same router. Therefore RMTI does not need to calculate with a given hop-count-limit like standard RIP does. As RMTI is a solution to the routing loop problem, overhead prone techniques like flooding of routing updates are not necessary. Investigations in our test environment showed that RMTI can converge faster than other distance vector routing protocols. In contrast to link-state protocols, RMTI allows the administration of local routing policies, which is a powerful method in order to impact the traffic density in the network. As RMTI does only a few additional loop tests in comparison to RIP, the runtime complexity grows still linearly with the number of subnets in a network domain.

Further investigations will be done to find rules for the optimization of the timer adjustments in order to maximize the benefit of the new RMTI technique. We will offer our new protocol as a Quagga daemon for Linux in order to invite readers to try out this new routing protocol. As RMTI can prevent all problems triggered by routing loops, it is a crucial improvement on the distance vector approach.

8. REFERENCES

[1] Ch. Steigner, H. Dickel, and T. Keupen, *RIP-MTI: A New Way to Cope with Routing Loops*, in Proceedings of the Seventh International Conference on Networking (ICN 2008), Cancun, Mexico , 2008.

[2] F. Bohdanowicz, H. Dickel, and Ch. Steigner, *Metric-based Topology Investigation*, in Proceedings of the Eighth International Conference on Networking (ICN 2009), Gosier, Guadeloupe/France, 2009.

[3] C. Cheng, R. Riley, S.P.R. Kumar, and J. J. Garcia-Luna-Aceves, *A loop-free extended bellman-ford routing protocol without bouncing effect*, ACM Sigc. Symp. Commun. Arch. and Prot., pp. 224-236, 1989.

[4] J. Dike, *User Mode Linux*, Prentice Hall, 2006.

[5] B. Rajagopalan and M. Faiman, *A new responsive distributed shortest-path routing algorithm*, ACM Sigcomm Symposium Commun. Arch. and Protocols, pp. 237-246, 1989.

[6] Francois, P. Bonaventure, O., *Avoiding Transient Loops During the Convergence of Link-State Routing Protocols*, Transactions on Networking, IEEE/ACM Volume: 15, Issue: 6, Dec. 2007

[7] J.J. Garcia-Luna-Aceves, *Loop Free Routing Using Diffusing Computations*, IEEE Transactions on Networking, 1993.

[8] Hengartner, U., Moon, S., Mortier R., and Diot, C., *Detection and Analysis of Routing Loops in Packet Traces*, Proc. of 2nd Internet Measurement Workshop (IMW 2002), Marseille, France, November 2002

[9] Kirill Levchenko, Geoffrey M. Voelker, Ramamohan Paturi, and Stefan Savage , *XL: An Efficient Network Routing Algorithm*, Proc. Sigcomm 2008, August, 2008.

[10] G. Malkin, *RIP Version 2*, RFC 2453, 1998, URL: http://tools.ietf.org/html/rfc2453, 05.08.2009.

[11] J. Moy, *OSPF Version 2*, RFC 2328, 1998, URL: http://tools.ietf.org/html/rfc2328, 05.08.2009.

[12] D. Pei, X. Zhao, D. Massey, and L. Zhang, *A Study of BGP Path Vector Route Looping Behavior*, IEEE International Conference on Distributed Computing Systems (ICDCS), March, 2004.

[13] C. E. Perkins, *Ad hoc networking*, Addison-Wesley, Amsterdam 2001.

[14] C. E. Perkins, E. Belding-Royer, S. Das, *Ad hoc On-Demand Distance Vector (AODV) Routing*, RFC 3561, 2003, URL: http://tools.ietf.org/html/rfc3561, 05.08.2009.

[15] Jian Qiu, Feng Wang and Lixin Gao, *BGP Rerouting Solutions for Transient Routing Failures and Loops*, in Proceedings of MILCOM, October, 2006.

[16] Quagga home page, http://www.quagga.net/, 05.08.2009.

[17] Y. Rekhter, T. Li, S. Hares, *A Border Gateway Protocol 4*, RFC 4271, 2006, URL: http://tools.ietf.org/html/rfc4271, 05.08.2009.

[18] A. Schmid and Ch. Steigner, *Avoiding Counting to Infinity in Distance Vector Routing*, Telecommunication Systems 19 (3-4): 497-514, March - April, 2002, Kluwer Academic Publishers.

[19] VNUML Project home page, http://www.dit.upm.es/vnumlwiki, 05.08.2009, Technical University of Madrid (UPM).

[20] OpenWRT Project home page, http://www.openwrt.org, 05.08.2009.

[21] Andrew S. Tanenbaum, *Computer Networks*, 3rd ed., Prentice Hall PTR, 1996, pp. 358-359

[22] Zifei Zhong, Ram Keralapura, Srihari Nelakuditi, Yinzhe Yu, Junling Wang, Chen-Nee Chuah and Sanghwan Lee, *Avoiding Transient Loops Through Interface-Specific Forwarding*, Transactions on Networking, IEEE/ACM, Volume: 15, Issue: 6, Dec. 2007 Transactions on Networking, Dec. 2007