Analyses and Improvements of Link Management Protocol for GMPLS-based Networks

Tie Luo
Beijing University of Posts and Telecommunications
Beijing, P.R. China
e-mail: tluo@glorisoft.com

G.S. Kuo
National Chengchi University
Taipei, Taiwan 116
e-mail: gskuo@ieee.org

Abstract – Generalized Multiprotocol Label Switching (GMPLS) is maturing to shape the next-generation mobile broadband IP networks, which will accommodate diverse technologies and various systems together. The Link Management Protocol (LMP), launched under the GMPLS context and being standardized by IETF, is designed for managing traffic-engineering (TE) links and verifying the reachability of control channels. A detailed study of the latest Internet-Draft on LMP has been conducted. Two important flaws in two of the four constituent procedures for LMP have been pointed out and improved by us in this paper. With regard to the link connectivity verification procedure, a batch-mode scheme is designed for enhancing its performance, scalability and flexibility, with theoretical analyses. In respect of the control channel management, a Privileged Hello Protocol is introduced to evade the dead-loop malfunction.

I. INTRODUCTION

Next-generation mobile broadband IP networks will consist of a large variety of technologies and network systems using Generalized MPLS (GMPLS) to dynamically provision resources and to provide network survivability. As a major feature, GMPLS separates signaling and routing function clearly from data forwarding function to provide a generic common control plane for diverse technologies that may be adopted.

For fulfilling the design goal of managing GMPLS-based networks, Link Management Protocol (LMP) is currently under the standardization process led by Internet Engineering Task Force (IETF). LMP runs between neighboring nodes and is used to manage traffic-engineering (TE) links and verify reachability of the control channel. It consists of four procedures, control channel management, link property correlation, link connectivity verification, and fault management.

Considerable endeavor has so far been made and the latest specification of LMP had recently been released by IETF [1]. Unfortunately, important flaws still exist behind the current protocol. The so-called Link Connectivity Verification procedure defined in [1] offers insufficient support for parallel processing, giving rise to problems concerning performance and scalability issues. With regard to the Control Channel Management procedure, a potential malfunction could occur on Hello messages initiated concurrently by both neighboring nodes, which can cause a dead loop as the consequence.

Corresponding solutions to the problems mentioned above are proposed in this paper respectively. A batch-mode link connectivity verification, serving as a substitute for the original counterpart, is proposed by us to address the performance and scalability issues, additionally to enhance flexibility as well. To avoid the potential dead loop, we propose a Privileged Hello Protocol, which completely eliminates the undesirable possibility using a simple scheme.

The rest paper is organized as follows. Section II analyzes the current LMP and identifies its imperfections. Our solutions are proposed with comparisons in Section III. Finally in Section IV, concluding remarks are made.

II. ANALYSIS OF LMP

A. Link Connectivity Verification

This procedure provides a mechanism that is used to verify the physical connectivity of the data links, and dynamically learns the TE link and Interface_Id associations as well. The procedure should be done initially when a TE link is established, and subsequently, on a periodic basis for all unallocated (free) data links of the TE link.

For performance purposes, data links are likely to be verified in parallel. Although the current design of LMP does not exclude this kind of situation, such verification has not so far been well considered and the support remains insufficiently provided by the protocol.

Generally, suppose a TE link between Node A and Node B consists of N unallocated data links among which N_f ones are broken. Node A initiates a parallel verification on the TE link by sending a BeginVerify message over a control channel. After Node B responding with a BeginVerifyAck message, Node A transmits N Test messages simultaneously, one per interface, over the N data links.

For the reason that Node B has no way to learn the neighbor's intention of performing verification in a parallel fashion, it has to deal with the matter in a common way. Upon receipt of each Test message, B maps the local Interface_ID to the received (remote) Interface_ID and marks this data link as UP; then it sends a TestStatusSuccess message over the control channel back to Node A indicating the health of this data link. On arrival of the message, Node A does the same mapping and marking job as B does and then replies with a TestStatusAck

message.

Since N_f messages are lost due to the failed data links, a timer, name it T_b, which is used to detect such failures, expires after an observation period (specified by the VerifyDeadInterval). Concluding that one of the data links has failed, B notifies A of this information using a TestStatusFailure message, meanwhile resetting the timer T_b. Subsequently, another period of VerifyDeadInterval elapses and the timer T_b expires again, just reproducing the message handshake. Such a situation will repeat in this way for N_f times until at last the session is terminated by Node A transmitting an EndVerify message, which normally occurs after A has received a sum of N TestStatusSuccess and TestStatusFailure messages. The above entire scenario is illustrated in Fig. 1.

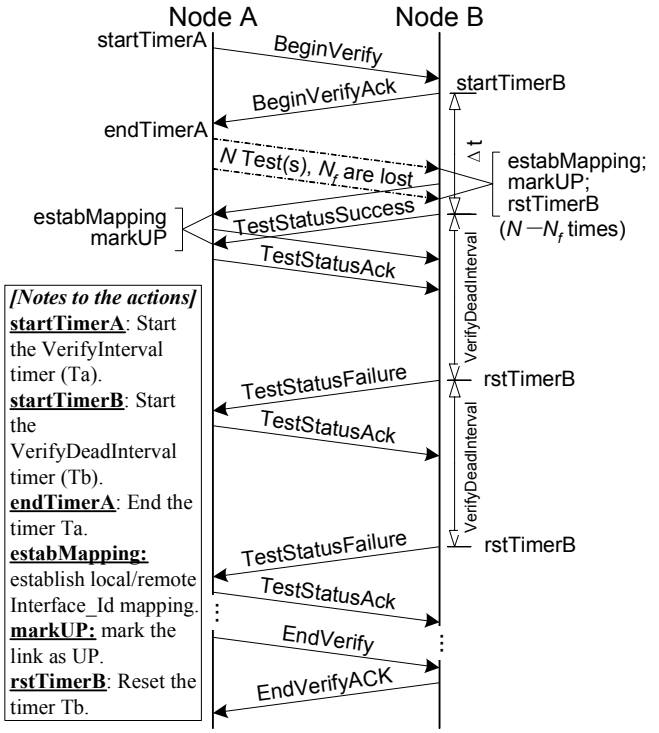


Fig. 1. A parallel verification scenario of the original LMP.

Such verification has an undesirable consequence in that discovering all the failures need spend a period of N_f • VerifyDeadInterval, whereas in fact a single expiration of T_b is sufficient enough for Node B to judge that the expected incoming N_f Test messages have all been lost. To the extreme, if the whole TE link is broken, i.e., N_f=N, we will have to tolerate such a long delay of N • VerifyDeadInterval that the performance of the procedure will degrade to that of serial processing.

A possible solution may suggest that this embarrassment can be overcome by Node A transmitting the EndVerify message immediately after acknowledging the first TestStatusFailure message originating from Node B. This EndVerify message, in addition to signifying the session can be terminated, carries an implication that all the rest data links without Test message coming out of have failed. Unfortunately, Node B cannot definitely recognize this additional meaning because, according to the protocol specification, the EndVerify message may be sent at any time when the initiating node, A, desires to end the Verify procedure.

Besides performance degradation, this undesirability also gives rise to a scalability issue on link bundling. As can be deduced from the previous illustration, a large TE link bundling numerous component links will risk encountering an intolerably long delay of verifying link connectivity, even performing a parallel verification improves little on condition that a considerable portion of those component links have failed. Therefore, a scalability limitation is potentially imposed upon GMPLS-based networks, hindering TE links from scaling up to accommodating a large quantity of component links.

B. Control Channel Management

Specifically designed for control channel management, the LMP Hello protocol can be used to maintain control channel connectivity between two adjacent nodes and to detect control channel failures once a control channel is activated.

According to the current protocol specification, after the parameter negotiation finished with a Config and a ConfigAck message, any of the two adjacent nodes can start sending Hello messages over a control channel. However, there is a possibility that both two nodes initiate a Hello message concurrently as a control channel is bi-directional. Each Hello message contains a RcvSeqNum indicating the sequence number of the last Hello message received from the adjacent node over this control channel, and the first message will have RcvSeqNum=0. In this case (see Fig. 2), because both of the two Hello messages contain RcvSeqNum=0, each node will regard the received message as an error packet with unexpected RcvSeqNum, and hence discard it and start to wait for a valid Hello message to come. Subsequently, after a HelloInterval timer expires, both of the nodes will do retransmission, but simply produce the same scene again. Continuously in this way, such a scene will be duplicated and, as the consequence, each node will be trapped in an endless loop.

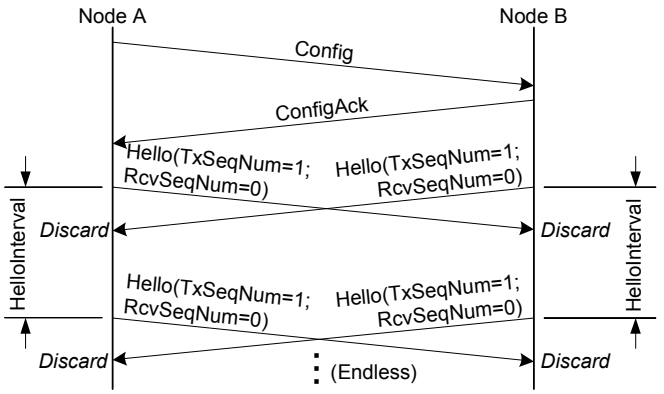
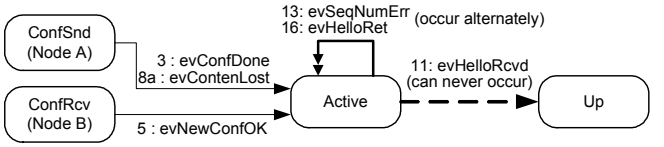


Fig. 2. The formation of the Hello dead loop.

The states and logics of operation of a LMP control channel, represented by a Finite State Machine (FSM), are partially drawn in Fig. 3 to illustrate the situation from another viewpoint. In this figure, event 13 and 16 occur alternately in persistence, constraining the control channel in the *Active* state and depriving it of the chance of entering the *Up* state.



[Notes to the states]

ConfSnd: In the parameter negotiation state, the node periodically sends a Config message, and is expecting the other side to reply with either a ConfigAck or ConfigNack message.

ConfRcv: In the parameter negotiation state, the node is waiting for acceptable configuration parameters from the remote side.

Active: In this state the node periodically sends a Hello message and is waiting to receive a valid Hello message. Once a valid Hello message is received, it can transition to the Up state.

Up: The CC is in an operational state.

[Notes to the events]

3 : evConfDone: A ConfigAck message has been received, acknowledging the Config parameters.

5: evNewConfOK: New Config message was received from neighbor and positively acknowledged.

8a : evContenLost: New Config message was received from neighbor at the same time a Config message was sent to the neighbor. The local node loses the contention, and the Config message is positively acknowledged.

11: evHelloRcvd: A Hello packet with expected SeqNum has been received.

13: evSeqNumErr: A Hello with unexpected SeqNum received and discarded.

16: evHelloRet: The HelloInterval timer has expired and a Hello packet is sent.

Fig. 3. A portion of control channel FSM (adapted from [1]).

III. Proposed Solutions

A. Batch-mode Link Connectivity Verification

We propose a batch-mode link connectivity verification procedure as a substitute for the current version described in the preceding section. This novel approach performs verification in batches, each of which contains a subset of all the component data-bearing links, as shown in Fig. 4.

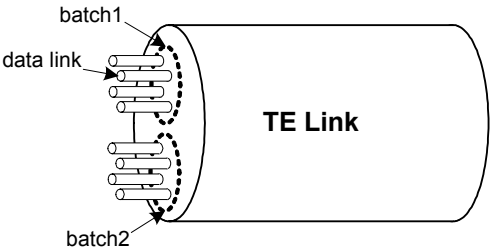


Fig. 4. The concept of a "batch".

Still consider two nodes and a TE link the same as before. Fig. 5 illustrates this new scenario. In the initialization stage of the verification, Node A sends a modified BeginVerify message which contains a new field informing the neighbor of the verify batch size, N_s. Then after receiving the acknowledgement from Node B, Node A transmits N_s Test messages over the first batch of N_s data links simultaneously, one message per link, to the adjacent node. At the remote side, once a Test message arrives at a certain interface, Node B will do the same mapping and marking job as in the scenario illustrated previously (Fig. 1) except that the timer resetting action is not needed now. If a VerifyDeadInterval timer expires before all the N_s anticipated TestStatusSuccess messages arrive, Node A recognizes that the rest corresponding data links in this batch have failed, so it marks them all as FAILED, and then sends a modified TestStatusFailure message to B indicating the link failures. To be noted here, the responsibility for judging data link failure is shifted from on Node B, as in the original protocol, to on Node A.

Multiple LOCAL_INTERFACE_ID objects are added to the TestStatusFailure message in order to enable the remote node to locate its own interfaces associating with the failed links had the local/remote Interface_Id mappings been established before. Such message modification can thereby detect failures earlier than the original version which is not able to find out these Interface_Ids until a entire round of test on all the component links has been completed. If the mapping is not available at that time, an optimization, fortunately, can solve the problem by testing the data links in a defined order known to both nodes, as pointed out in [1]. Besides, this modification has another advantage that allows all failures belonging to one batch to be reported using just a single message, thus accelerating the processing and saving the bandwidth.

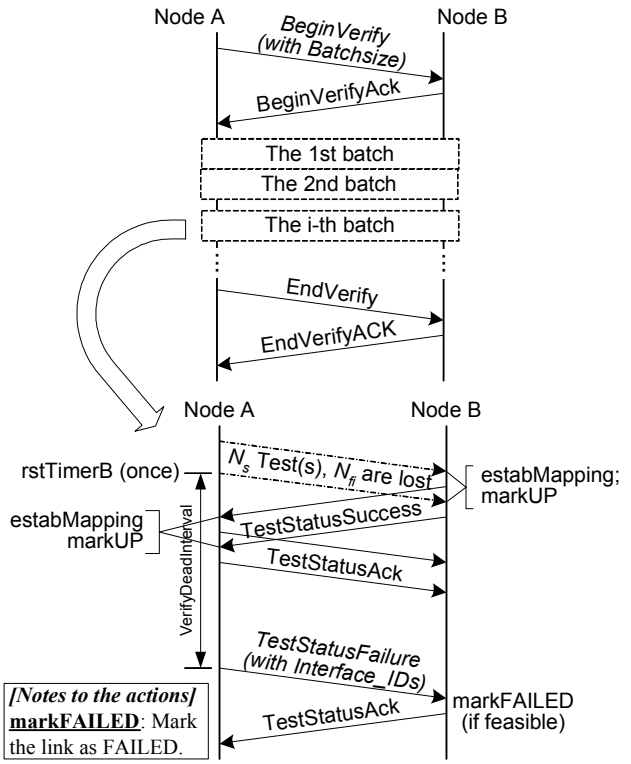


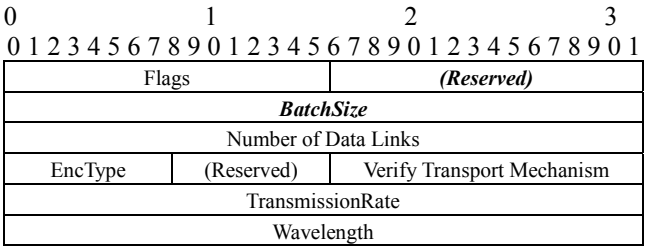
Fig. 5. A scenario of the batch-mode link connectivity verification.

Of course, the last batch may contain less than N_s links as N is not always a multiple of the batch size. It does not matter, however, because both nodes can be cognizant of this fact via calculation.

To be particularly pointed out, this novel verify scheme encompasses both serial processing simply by setting the batch size to *1*, and parallel processing by setting the size to N.

The new procedure requires the following revisions to the original LMP:

- Remove the VerifyInterval (the interval at which the Test messages will be sent) field from the BeginVerify message and add a BatchSize field. (Fig. 6)
- Remove the VerifyDeadInterval field from the BeginVerifyAck message, and change this interval into a local configuration parameter. (Fig. 7)
- Add LOCAL_INTERFACE_ID objects to the TestStatusFailure message. (Fig. 8)



BatchSize: 32 bits

This is the number of data links that will be verified in each batch. (Refer to [1] for description of other fields.)

Fig. 6. The new BEGIN_VERIFY object of the BeginVerify message.

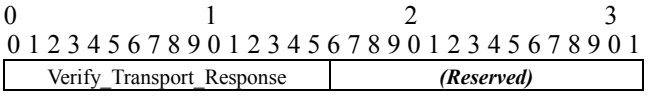


Fig. 7. The new BEGIN_VERIFY_ACK object of the BeginVerifyAck message.

<TestStatusFailure Message>::=<Common Header><MESSAGE_ID>
<LOCAL_INTERFACE_ID>[<LOCAL_INTERFACE_ID>…]
<VERIFY_ID >

Fig. 8. The new TestStatusFailure message.

Benefits, as listed below, can be achieved from this innovative batch-mode verify scheme:

- Unified architecture – Serial, parallel, and even mixed verification ($1<N_s<N$; serial inter-batch processing plus parallel intra-batch processing) can all be well supported by the new mechanism; different processing specific to each type is now unified into a common architecture.
- Promoted performance – The overall efficiency of the verification is remarkably improved. Mathematical calculation is presented below to demonstrate this point.

  The variables used are defined in Table 1.

TABLE 1. VARIABLES DEFINITION

Variable name	Description
N	The number of all the data links to be verified
N_f	The number of all the failed data links
N_{fi}	The number of the failed data links in the *i-th* batch
N_s	The batch size
N_b	The number of the batches ($N_b = \lceil N / N_s \rceil$, in which the expression $\lceil x \rceil$ means the nearest integer not less than x.)
T	The time span of verifying all the data links once (excluding the BeginVerify and the EndVerify stage)
T_{Bi}	The time span of verifying the *i-th* batch ($i=1, 2, 3, \ldots, N_b$)
T_{dl}	Average delay of a Test message transmitting over a data link
T_{cc}	Average delay of a message transmitting over a control channel
T_{proc}	Average time for processing a message on a node
T_{dead}	The interval of VerifyDeadInterval

The following premise is also taken:

$$T_{dead} > T_{dl} + T_{cc} + T_{proc} \quad (1)$$

1) *Use the original LMP*

– Serial verification

The first equation is derived from Fig. 9:

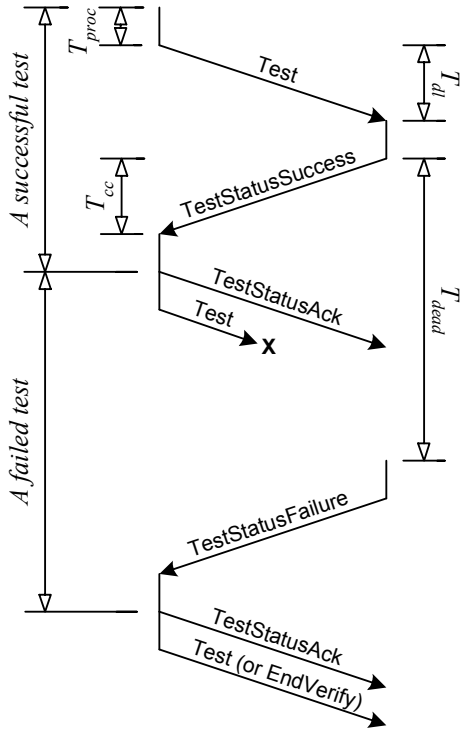


Fig. 9. Serial processing of the original LMP.

$$T = (N - N_f) \bullet (T_{dl} + T_{cc} + 3\,T_{proc}) + N_f \bullet (T_{dead} + T_{proc}) = (N - N_f) \bullet (T_{dl} + T_{cc}) + N_f \bullet T_{dead} + (3N - 2N_f) \bullet T_{proc} \quad (2)$$

– Parallel verification

The reference diagram is shown as Fig. 10.

If $0 \leqslant N_f < N$,
$T = T_{dl} + T_{cc} + (N+2) \bullet T_{proc} + N_f \bullet (T_{dead} + T_{proc})$.
(Assuming the last Test message arrives successfully.)

If $N_f = N$,
$T = N \bullet T_{dead} + (N+1) \bullet T_{proc}$.

Thus

$$T = \begin{cases} T_{dl} + T_{cc} + N_f \bullet T_{dead} + (N+N_f+2) \bullet T_{proc}, & 0 \leqslant N_f < N \\ N \bullet T_{dead} + (N+1) \bullet T_{proc}, & N_f = N \end{cases} \quad (3)$$

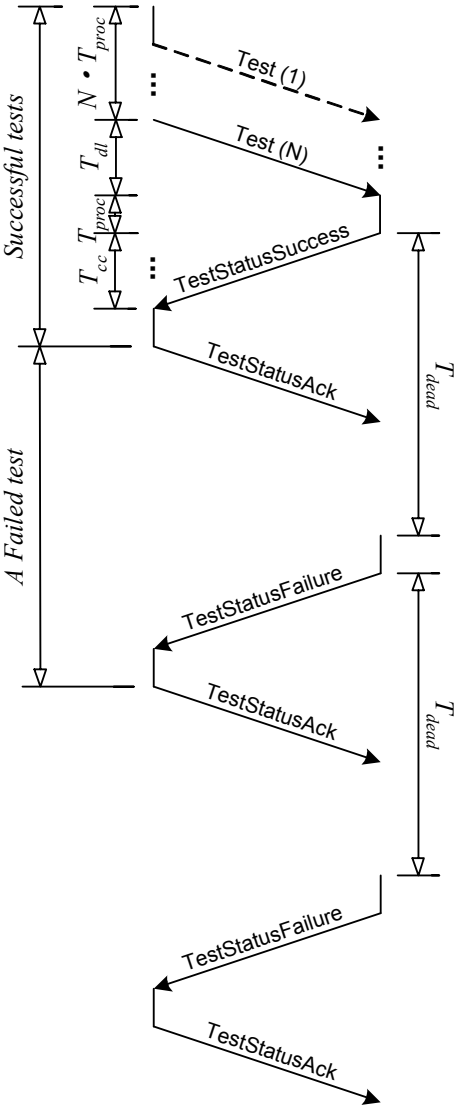


Fig. 10. Parallel processing of the original LMP.

2) *Use the proposed LMP*

Fig. 11 serves as the reference diagram.

$T = \sum_{i=1}^{N_b} T_{Bi}$, in which

$$T_{Bi} = \begin{cases} T_{dead} + 2T_{cc} + (N_s+2) \bullet T_{proc}, & 0 < N_{fi} \leqslant N_s, \\ T_{dl} + T_{cc} + (N_s+2) \bullet T_{proc}, & N_{fi} = 0 \end{cases}$$

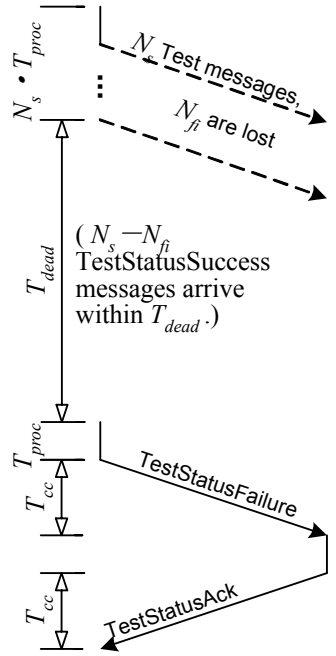


Fig. 11. Batch-mode verification (the *i-th* batch) of the proposed LMP.

As two special cases, $N_s=1$ and $N_s=N$ are considered respectively:

– $N_s=1$ (serial verification)

$$\because T_{Bi} = \begin{cases} T_{dead} + 2T_{cc} + 3T_{proc}, & N_{fi}=1 \\ T_{dl} + T_{cc} + 3T_{proc}, & N_{fi}=0 \end{cases}$$

$N_b=N$,

$$\therefore T = \sum_{i=1}^{N_b} T_{Bi}$$

$$= N_f \cdot (T_{dead} + 2T_{cc} + 3\,T_{proc}) + (N - N_f) \cdot (T_{dl} + T_{cc} + 3\,T_{proc})$$
$$= (N - N_f) \cdot T_{dl} + (N+N_f) \cdot T_{cc} + N_f \cdot T_{dead} + 3N \cdot T_{proc} \quad (4)$$

– $N_s=N$ (parallel verification)

$$T = \begin{cases} T_{dead} + 2T_{cc} + (N+2) \cdot T_{proc}, & 0<N_f \leq N \\ T_{dl} + T_{cc} + (N+2) \cdot T_{proc}, & N_f=0 \end{cases} \quad (5)$$

3) *Comparison*

Fig. 12 illustrates the relationship between T and N_f for the two different versions of LMP with parameters defined in Table 2.

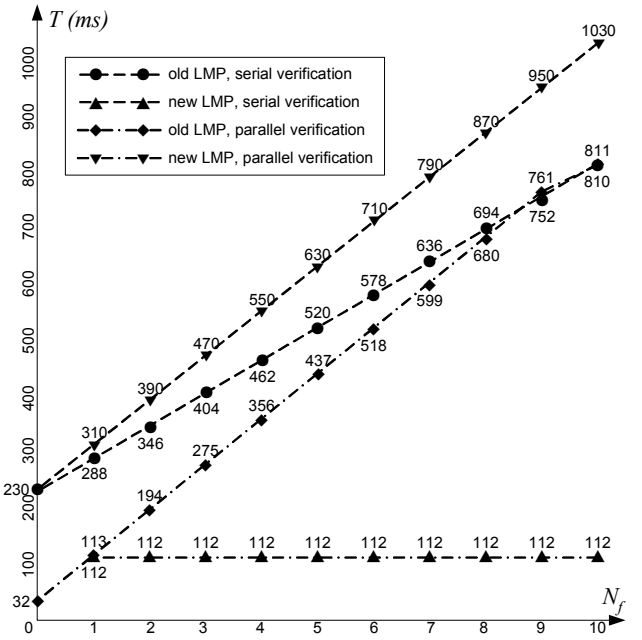


Fig. 12. The relationship between T and N_f.

TABLE 2. PARAMETERS VALUE ASSIGNMENT

N	T_{dl}	T_{cc}	T_{dead}	T_{proc}
10	10 ms	10 ms	80 ms	1 ms

Conclusion can be safely drawn as follows (use T_{old} to stand for T of the original LMP, and T_{new} for T of the proposed LMP.):

- In the parallel situation, usually we have $T_{new} < T_{old}$, and the difference between them, $T_{old} - T_{new}$, will be augmented sharply as the number of failed data links (N_f) increases.
- In the serial situation, T_{new} is only slightly greater than T_{old}, and it is crucial to be aware that only when verifying a small-scale TE link (e.g. $N \leqslant 10$) could the batch-mode scheme regress to choosing serial processing. Therefore the difference, $T_{old} - T_{new}$, can be confined to a very narrow range.
- In other situations ($1<N_s<N$), the performance of the batch-mode link verification is expected to vary between the corresponding two boundaries (the uppermost curve and the lowermost curve). From area of the divisions we can deduce that the probability of $T_{new}<T_{old}$ will be much bigger than that of $T_{new}>T_{old}$.

- Mitigated scalability limitation – From equation (2), we can learn that in parallel verification using the old LMP, T is remarkably influenced by N_f and N. It may grow much larger as N goes up. On the contrary, using the improved version, we know from equation (5) that T stays perfectly stable as N increases (T_{proc} is usually negligible), and it is even immutable to N_f. A TE link, therefore, is capable of scaling up to a large bundle with much more component links accommodated.
- Enhanced flexibility – Using the proposed LMP, not

only is a network node able to be configured to use an appropriate batch size according to a specific TE link, but it is also allowed to adjust the value dynamically in response to the size of a TE link varying. This flexibility is especially useful when considering implementation issue. For instance, applying multithreading technique to implement LMP may be a good choice but usually only a limited number of active threads can be supported on a computer, or a specific size of thread pool is able to achieve better performance. On some platforms, multithreading may be not even supported. In all the cases, the capability of adjusting to a (near-) optimal batch size will facilitate desirable network provision, configuration and operation.

- Simplified operation – The VerifyInterval timer is no longer needed since it is the initiating node (Node A in the previous example) that has the duty of detecting link failure, so this node is able to control the progress without relying on a timer. Associated tasks are thereby got rid of and operation of the procedure is simplified.

B. Privileged Hello Protocol

Named as Privileged Hello Protocol, our solution is designated for solving the problem of creating a dead loop during the fast keep-alive phase. The central idea of this protocol is straightforward: only one of the two adjacent nodes is privileged to initiate the first Hello message.

The mechanism is derived from a scheme that has already been developed to avoid ambiguities produced by both two nodes initiating the Hello parameter negotiation (this phase will activate a control channel) at the same time. In the Privileged Hello Protocol, the node that is granted the privilege to transmit the first Hello message is the very node that successfully initiates the configuration procedure – either the unique node that sends the Config message, or one of the two nodes who wins the contention with higher Node_Id if both of them send the Config simultaneously. In other words, only the node that has received a ConfigAck message is permitted to say Hello first, and the other node is responsible for replying the greetings. With this asymmetric mechanism used, chances for creating the dead loop are completely eliminated, and the design goal of the original Hello protocol, to maintain control channel connectivity and detect control channel failures, is still fulfilled without any impairment.

IV. Conclusions And Future Work

This paper performs in-depth analyses on LMP defined by IETF [1] and points out important weak points lying behind two of its four constituent procedures. Corresponding solutions are also proposed respectively to improve LMP. A batch-mode link connectivity verification is first proposed for addressing the performance and scalability issues with flexibility enhancement. Next, a privileged Hello protocol is proposed to completely eliminate the dead-loop malfunction.

It is also necessary to conduct computer simulations and to establish test-beds for carrying out comprehensive experiments on LMP, our proposed improvements, and further new findings, since LMP, unquestionably, bears considerable significance to GMPLS, which is to play a crucial role in the next-generation mobile broadband IP networks.

References

[1] J. P. Lang, et al., “Link Management Protocol (LMP),” Internet draft, draft-ietf-ccamp-lmp-07.txt, November 2002, work in progress.

[2] E. Mannie, et al., “Generalized Multi-Protocol Label Switching (GMPLS) Architecture,” Internet draft, draft-ietf-ccamp-gmpls-architecture-04.txt, February 2003, work in progress.

[3] A. Fredette, et al., "Link Management Protocol (LMP) for DWDM Optical Line Systems," Internet draft, draft-ietf-ccamp-lmp-wdm-01.txt, September 2002, work in progress.

[4] John Drake, et al., “Control Channel Bootstrap for Link Management Protocol,” Internet draft, draft-lang-ccamp-lmp-bootstrap-02.txt, December 2002, work in progress.

[5] G. Bernstein, et al., “Link Management Protocol Update,” Internet draft, draft-everdingen-ccamp-lmp-update-00.txt, June 2002, work in progress.

[6] D. Tappan, et al., "LMP Extensions for Link discovery Using Loss of Light," Internet draft, draft-rbradfor-ccamp-lmp-lol-00.txt, October 2002, work in progress.

[7] A. Banerjee, et al., “Generalized Multiprotocol Label Switching: An Overview of Routing and Management Enhancements,” *IEEE Comm. Mag.*, pp. 144-150, January 2001.

[8] A. Banerjee, et al., “Generalized Multiprotocol Label Switching: An Overview of Signaling Enhancements and Recovery Techniques,” *IEEE Comm. Mag.*, pp. 144-151, July 2001.

[9] A. Shami, et al., “Performance Evaluation of Two GMPLS-Based Distributed Control and Management Protocols for Dynamic Lightpath Provisioning in Future IP Networks,” *Proc. IEEE ICC 2002*.

[10] T. W. Um, et al., “Signaling and Control Procedures Using Generalized MPLS Protocol for IP over an Optical Network,” ETRI Journal, vol. 24, no. 2, April 2002.

[11] M. Z. Hasan, “Multi-Technology, Multi-Vendor Optical Network Provisioning – Management Plane Perspective,” OIF, Dallas, November 2001.