Optica Applicata, Vol. XXXV, No. 2, 2005

Design of a test suite for the jumpstart just-in-time signaling protocol

A. HALIM ZAIM

Istanbul University, Faculty of Engineering, Department of Computer Engineering, Avcilar, Istanbul, Turkey; e-mail: ahzaim@istanbul.edu.tr

We present a two step design of the just-in-time (JIT) signaling scheme running over a core dWDM network which utilizes optical burst switches (OBS). In the first step, we apply an eight-tuple extended finite state machine (EFSM) model to formally specify the protocol. Using the EFSM model, we define the communication between a source client node and a destination client node through an ingress and an intermediate switch. The communication between the EFSMs is handled through messages. In the second step, we generate unique input–output sequences. We work on on-the-fly unicast connection over a prototype testbed.

Keywords: optical burst switching (OBS), just-in-time (JIT) signaling, optical communication.

1. Introduction

The protocol design process requires a formal definition of the protocol and some way of defining output sequences for a given set of inputs [1]. More information about different protocol test techniques can be found in [2–4].

In recent years, a great change in the protocol design is observed in telecommunication industry. Instead of the traditional design cycle, which includes a three-step process consisting of a high level design, low level design and coding and testing, a more formal design approach has been developed. The formal design approach uses methods that help the designer verify the correctness of the design decisions as they are made. For more information on formal design approaches, refer to [5–8].

Extended finite state machine (EFSM) approach is fully expressive and particularly useful as a means of describing communication protocols. EFSM-based techniques can be applied in telecommunication easier than most other approaches, and are better suited to assist in the followup implementations. Therefore, in this study we used an EFSM based description model.

Jumpstart signaling protocol was first introduced in [9]. The signaling architecture is based on wavelength routing and burst switching. Signaling is just-in-time (JIT), indicating that signaling messages travel slightly ahead of the data they describe.

Signaling is out of band, with signaling packets undergoing electro-optical conversion at every hop. Data is opaque to network entities and travels through the network in bursts of varying durations, each burst preceded by its own signaling message.

Optical burst-switching (OBS) is a promising direction of research and development in wavelength-routed core WDM networks. Coupled with out-of-band signaling it promises to deliver a transparent all-optical architecture, capable of transporting digital and analog data, regardless of format. JIT signaling approaches to OBS have been previously studied in the literature [10–14]. These approaches are characterized by the fact that the signaling messages are sent just ahead of the data to inform the intermediate switches. The common thread is the elimination of the round -trip waiting time before the information is transmitted (the so-called tell-and-go approach): the switching elements inside the switches are configured for the incoming burst as soon as the first signaling message announcing the burst is received. The variations on the signaling schemes mainly differ in how soon before the burst arrival and how soon after its departure the switching elements are made available to route other bursts through the use of the combination of signaling messages and timers.

The organization of the paper is as follows: in Sec. 2, jumpstart signaling protocol is explained briefly. In Section 3, we define reachability trees and generate UIO sequences. After explaining an implementation assessment model, Sec. 4 concludes our paper.

2. Jumpstart just-in-time signaling protocol

Jumpstart signaling uses a link-unique identifier (or label) for each message, which upon emergence on the other end of the link can be cached and mapped to a new identifier or label on the exit link. The first message in a signaling flow (SESSION DECLARATION or SETUP) serves the purpose of setting up a label-switched path, which all further messages in forward and reverse direction follow. That is, this on-the-fly setup of a label switched path is the main difference between MPLS or ATM and our approach. Another difference worth noting is that in MPLS, labels are distributed upstream, in the reverse direction of the path prior to path being used. In our case we need to setup the label-switched path in the forward direction. In addition, we must setup the reverse path at the same time.

T a b l e 1. Signaling protocol functions.

Session Declaration	Announce the connection to the network
Path Setup	Configure resources needed to setup an all-optical path from source to destination
Data Transmission	Inform intermediate switches burst arrival time and length
State Maintenance	Keep up the necessary state information to maintain the connection
Path Teardown	Release resources taken up to maintain the lightpath for the connection

The basic signaling protocol for a JIT OBS network described in this section only addresses the connection setup procedures.

Depending on the type of the connection being setup, the signaling protocol may need to perform several functions, all described in Tab. 1. These phases can be accomplished by the signaling protocol in a different fashion, depending on the assumptions made about the network: the reliability of individual links, scheduling capabilities of the switches and other factors.

Although we propose to use two types of connection setup in the jumpstart protocol called, explicit_setup_and_explicit_teardown and explicit_setup_and_estimated_teardown, in the testbed, only explicit_setup_and_explicit_teardown is implemented. The details about the differences between these two signaling schemes can be found in [9].

2.1. Connection phases

Each connection in our OBS network goes through a number of well-defined phases as described in [9]. This paper concentrates on a unicast case. Unicast connections have all of these phases; however, some of them are collapsed into a single step. For example for short bursts the Setup message serves to:

– announce the session to the network (Session Declaration);
– setup the path of the session (Path Setup);
– announce the arrival of the burst (Data Transmission).

This way the Setup message combines the three phases, which are followed by either an explicit or implicit session Release. Path teardown phase may be explicit (if explicit teardown with a Release message is used) or implicit (if estimated teardown with a Timeout message is used)*. These simple connections lack State Maintenance phase due to their short-lived nature. This phase is intended for long-lived bursts that require the "keep-alive" message.

2.2. Unicast message flows

The next subsection presents the flow of signaling messages for unicast connections for on-the-fly short bursts.

2.2.1. On-the-fly unicast signaling flows

These connections combine the Session Declaration, Path Setup, and Data Transmission phases into a single Setup message.

The message flows for short bursts and lightpaths are presented in Fig. 1. The presence of the Release message at the end of each connection is dictated by the type of the connection (explicit *vs*. timed teardown).

Regardless of the type of the connection, it is initiated with a Setup message sent by the originator of the burst to its ingress switch. The ingress switch consults with delay estimation mechanism based on the destination address and returns the updated

*In this study we only take into account explicit teardown.

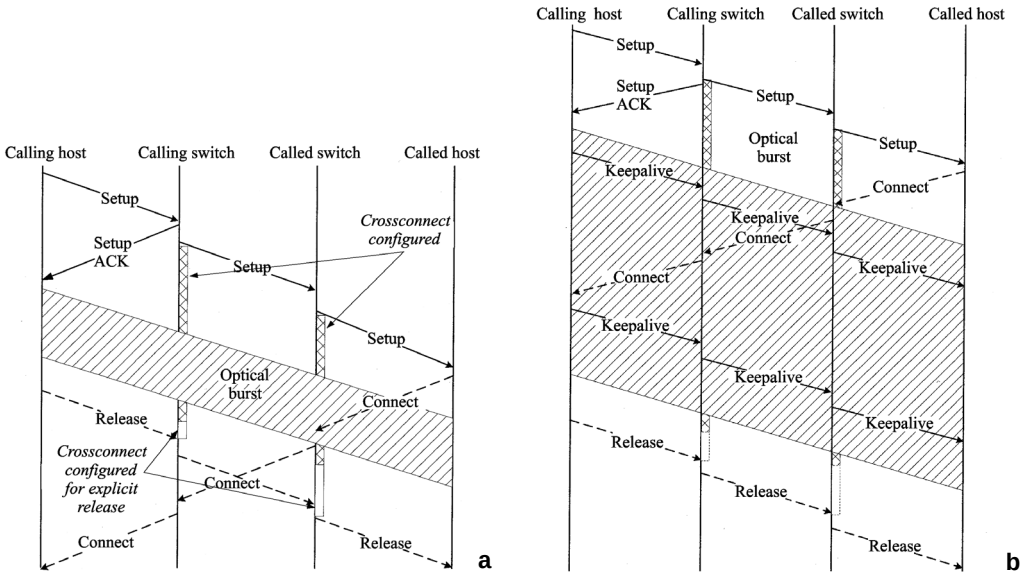

Fig. 1. Supported connection types: short burst (**a**), lightpath (**b**).

delay information to the originator by using a Setup Ack message, at the same time acknowledging the receipt of the Setup message by the network. The Setup Ack message also informs the originating node which channel/wavelength to use when sending the data burst.

The originator waits for the required balance of time left based on its knowledge of the round-trip time to the ingress switch, and then sends the burst on the indicated wavelength. The Setup message at the same time is traveling across the network, informing the switches on the path of the burst arrival. If no blocking occurs on the path, the Setup message eventually reaches the destination node, which then receives the incoming burst shortly thereafter.

Upon the receipt of the Setup message, the destination node may choose to send a Connect message acknowledging the successful connection (indeed, the receipt of the Setup by the destination only guarantees that the connection has been established; it does not guarantee its successful completion, since a connection may be preempted somewhere along the path by a higher-priority connection).

For long-lived bursts, the Keepalive message maintains the state of the connection, preventing it from timing out. Especially for explicit teardown, where a connection is not closed until a Release message is received, Keepalive message is used to notify the aliveness of the source. Otherwise, in case that the source is dead, if there is no Keepalive mechanism, the connection will wait for a Release forever wasting the limited crossconnect resources. However, with Keepalive mechanism, if the source does not send a Keepalive message during a specified time, a timeout occurs and the connection is closed.

One message type not mentioned above is sent if any type of failure is detected during setup or maintenance phase of the connection. This message is called Failure. It is sent to the originator of the connection and it carries with it the cause of failure, including blocking, preemption by a higher-priority connection, lack of route to a host, refusal by destination, *etc*.

3. Protocol testing

Protocol testing can be classified into three subcategories, each subcategory defining one part of the testing process:

– protocol validation,
– conformance testing,
– implementation assessment.

Protocol validation deals with formal protocol description. Its aim is to find logical errors on formal protocol specifications. Conformance testing is used to check consistency of the protocol implementation with its specifications. That is, whether the external behavior of a given implementation of a protocol is equivalent to its formal specifications. From that point of view, it is obvious that, if there is a logical error on

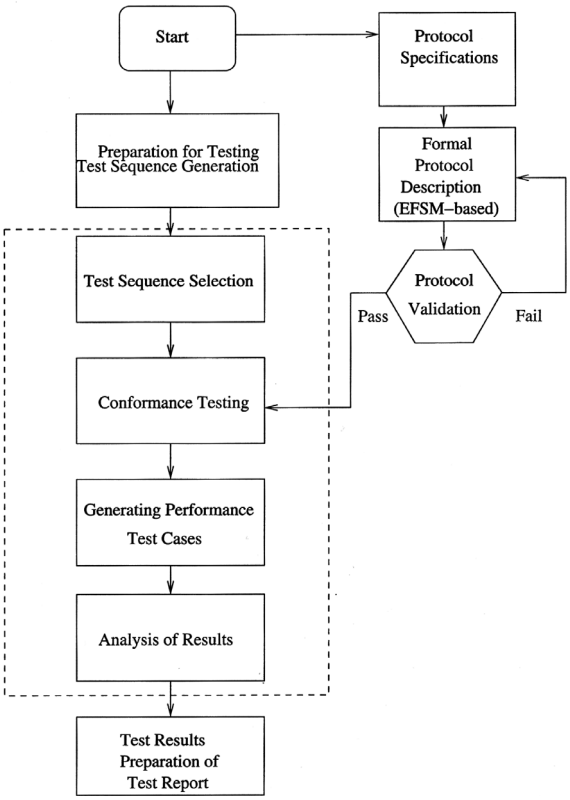

Fig. 2. Protocol test process.

the protocol specifications, the conformance test will not catch an error unless there is another error on conformance test generation. Implementation assessment, on the other hand, is more general. It is mostly related to properties of the protocol that are not part of the protocol specifications such as how the implementation reacts to unexpected (invalid) user commands, how many simultaneous connections can be supported, or evaluation of the performance of the implementation compared with analytical results related with the given protocol and with competitor protocols. Therefore, in order to assess the implementation without error, the implementation environment should also be taken into account. In other words, the implementation can not be assessed in isolation.

The test process can be illustrated using Fig. 2.

3.1. Protocol validation

The aim of the protocol validation is to define the protocol specifications in a formal language and then test the logical consistency of the protocol description. The first step of the protocol validation has been performed in [16] by defining the verbal protocol specifications with an EFSM model. The next step on the validation process is to check for logical consistencies. The classical validation technique is to make a reachability analysis. There are three approaches to reachability analysis: full search (for systems up to 10^5 states), controlled partial search (for systems up to 10^8 states) and random simulation (for larger systems). Jumpstart protocol has approximately 400 states as a compound system machine. Therefore, a full search using an exhaustive search algorithm is the best candidate. The result of this search will create all valid sequences, eliminating erroneous sequences. However, unlike the algorithms used on communicating finite state machines, in this section, using decomposition technique, we will apply the full search technique to each individual state machine independently, because the EFSMs do not make state transitions at the same time. The rest of this section is devoted to the reachability analysis of individual state machines for the source, destination, ingress switch and intermediate switch.

3.1.1. Reachability analysis for the source

Source state diagram has five states, and eleven messages. In this subsection, a tree structure is used to show the reachability starting from the initial state IDLE as the root of the tree. The reachability tree is shown in Fig. 3.

This tree can now be used to create valid input sequences following the branches. Once valid input sequences are generated, we define valid output sequence for the given input sequence. Due to space limitations, we omit the input and output sequences, but provide unique input/output (UIO) sequences in Sec. 3.2.1.

3.1.2. Reachability analysis for the destination

Destination state diagram has three states, and eleven messages. In this subsection, a tree structure is used to show the reachability starting from the initial state IDLE as

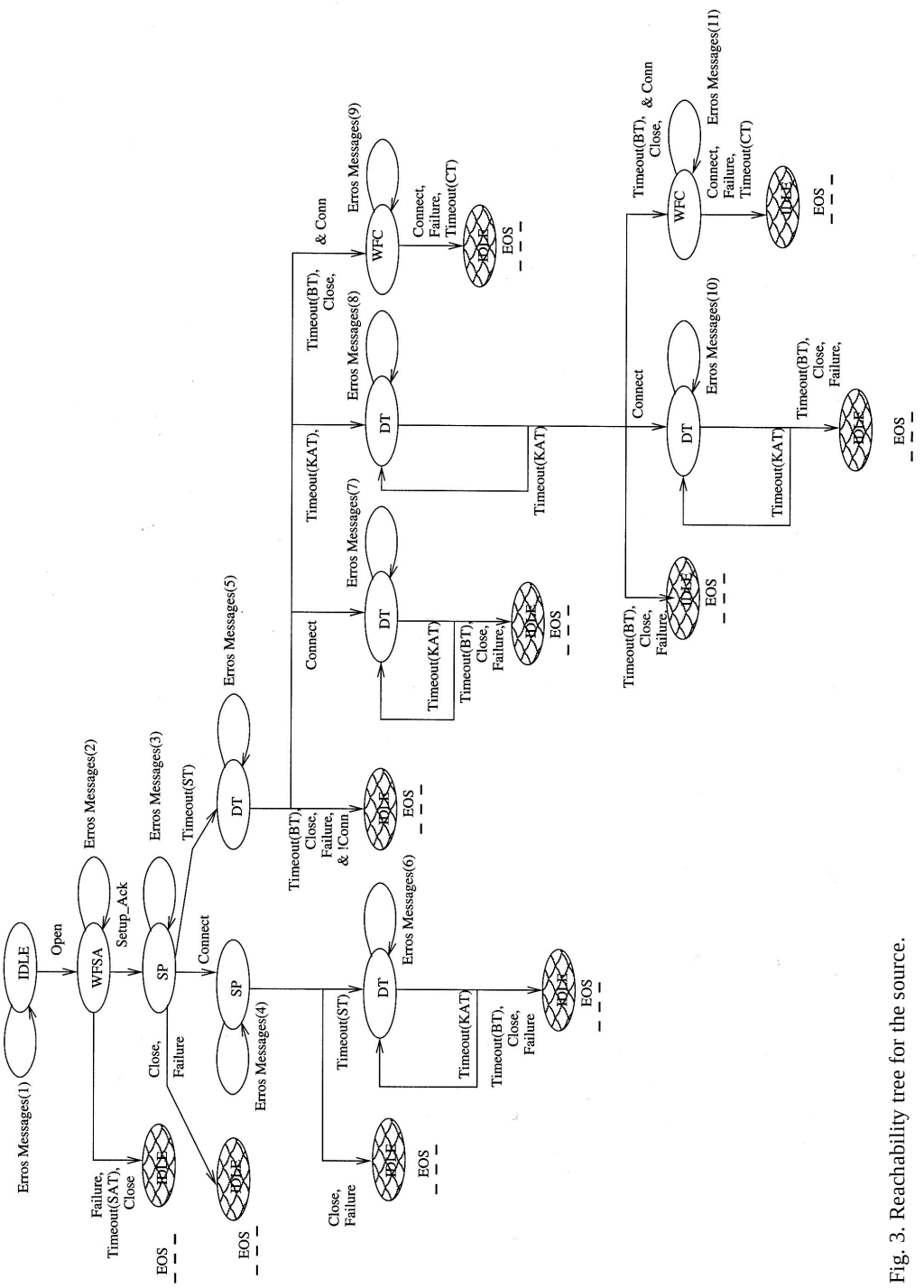

Fig. 3. Reachability tree for the source.

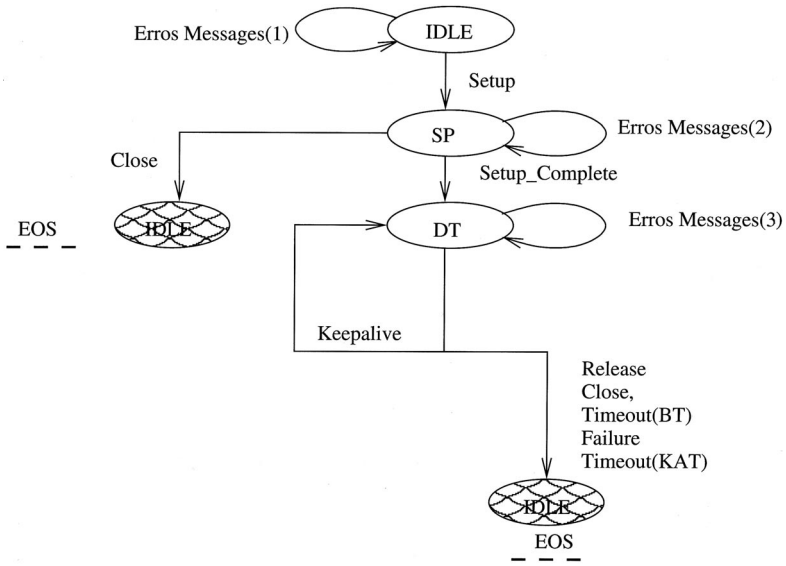

Fig. 4. Reachability tree for the destination.

the root of the tree. The reachability tree is shown in Fig. 4. This tree is again used to create valid input and output sequences.

3.1.3. Reachability analysis for the switch

Switch state diagram has three states, and seven messages. In this subsection, a tree structure is used to show the reachability starting from the initial state IDLE as the root of the tree. The reachability tree is shown in Fig. 5.

3.2. Conformance testing

The conformance test for the JIT protocol can be performed as illustrated in Fig. 6. There will be two tester units, Peer layer tester (PLT) and upper layer tester (ULT) to send input messages generated in Sec. 3.1. The reactions of implementation under test (IUT) to these messages or in other words, the valid output messages will be observed in observation channel 1 (OC1) and observation channel 2 (OC2). In order to make the testing easier, during the conformance test, the valid output sequences will also include the channel in which each message is expected. The rest of this chapter is devoted to valid output sequences generated for each element mentioned in Sec. 3.1.

3.2.1. Unique input/output (UIO) sequences

Lai defines a unique input/output (UIO) sequence in [3] as “a UIO sequence for a state is an input/output behavior that is not exhibited by any other state”. The protocol testing with UIO is performed by feeding the system with a specified input sequence and

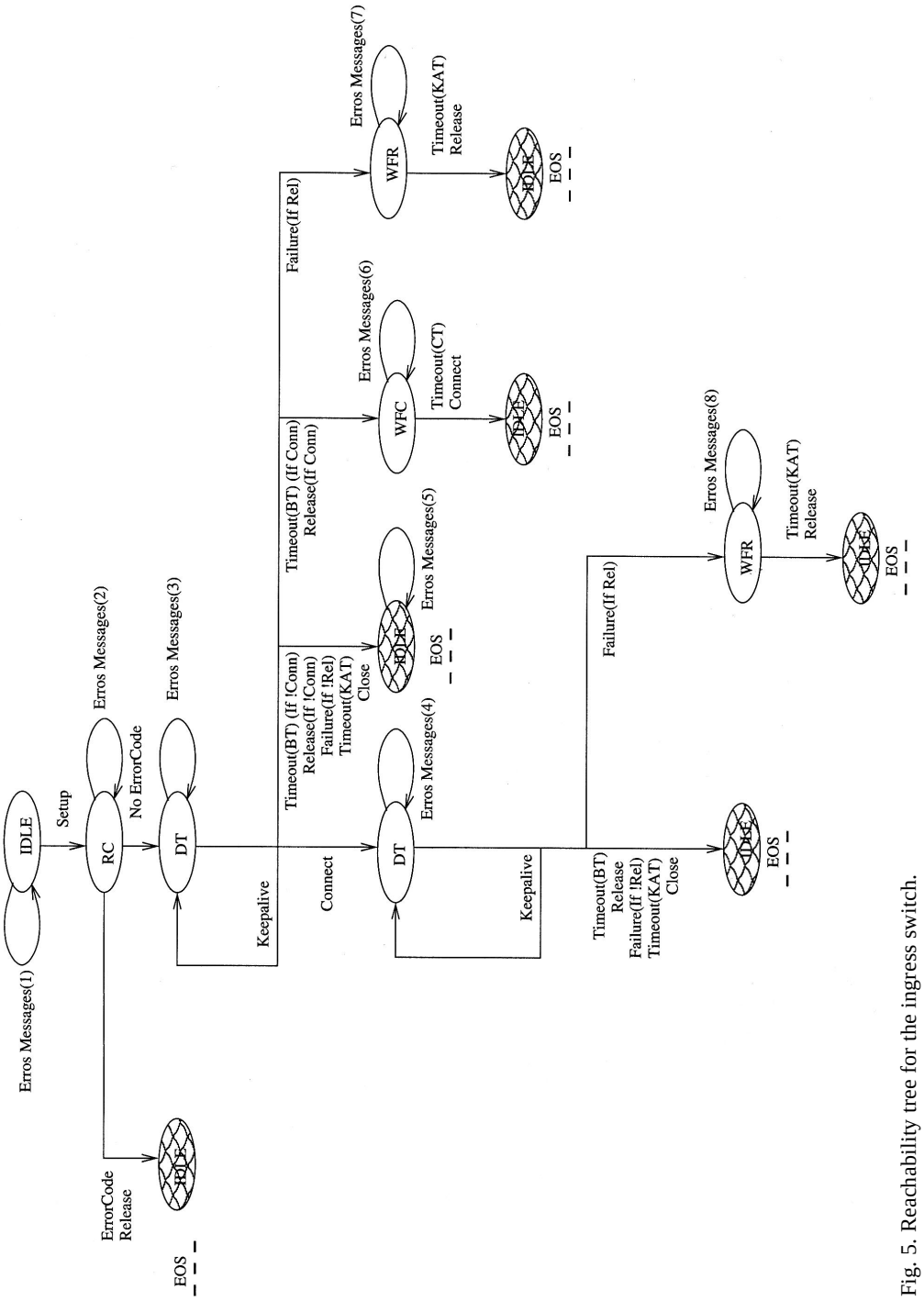

Fig. 5. Reachability tree for the ingress switch.

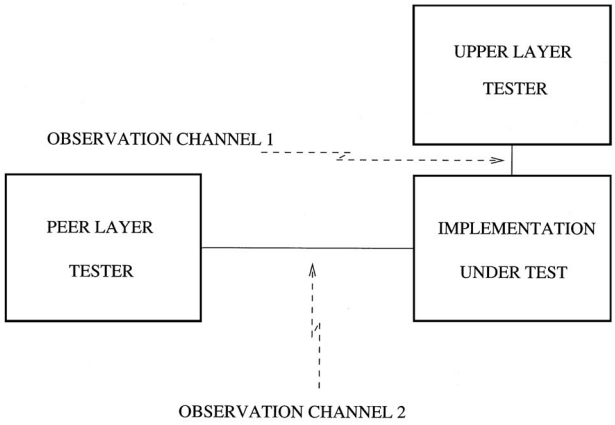

Fig. 6. Protocol testing architecture.

observe the output sequence. If the output sequence is the same with any UIO sequence defined, then this information gives us the initial state of the state machine.

Unique input/output sequences and distinguishing sequences are not the same. In a distinguishing sequence, the input sequence is the same for all states. The states can be distinguished by the output sequence generated for the same input sequences coming to each state. However, in an UIO sequence, usually, each state has a different input

Table 2. UIO sequence for the source.

State	UIO sequence
IDLE	Open/Setup
WAIT_FOR_SETUP_ACK	Timeout(SA_Timer)/Release
SETUP_PROCEEDING	Timeout(Setup_Timer)/Clear_To_Send
DATA_TRANSMISSION	Timeout(KA_Timer)/Keepalive
WAIT_FOR_CONNECT	Timeout(Conn_Timer)/Connection_Failure

Table 3. UIO sequence for the destination.

State	UIO Sequence
IDLE	Setup/Open
SETUP_PROCEEDING	Setup_Complete/λ
DATA_TRANSMISSION	Timeout(KA_Timer)/Transmission_Failure

Table 4. UIO sequence for the switch.

State	UIO Sequence
IDLE	Setup/λ
RUNNING_CHECKS	Release/Failure
DATA_TRANSMISSION	Release/Release

sequence. For a given state *s*, the input part is applied to the state machine and the output is observed. If it is the same with the expected output sequence, then it can be concluded that the state machine is in state *s*. In Tables 2–4, for each state machine (source, destination and switch), an UIO sequence is generated, respectively, where λ denotes a null output.

3.3. Implementation assessment

Implementation assessment mostly consists in generating test scenarios to assess the performance of the implementation. For our system under implementation, the system structure is shown in Fig. 7.

As shown in Fig. 7, our initial implementation consists of two end nodes, Client 1 and Client 2, and two OBS switches using just-in-time protocol accelerator cards (JITPAC). Although each switch has four ports, only one port will be tested in this first implementation. Each port can carry up to eight wavelengths. Once the system architecture is given, we should mention the necessary input parameters and the output parameters that will be observed as the outcome of these tests.

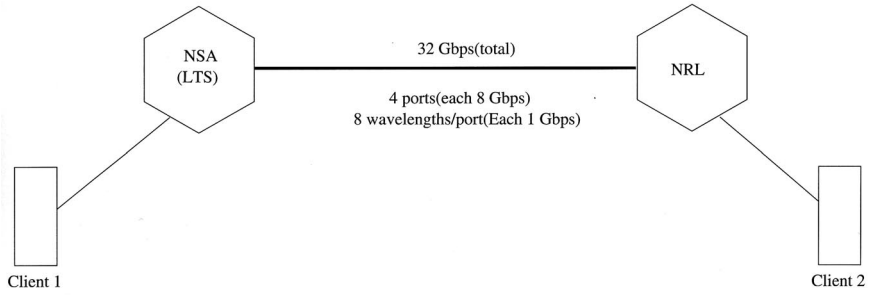

Fig. 7. Testbed architecture.

This system is a prototype system, and as there are only two nodes, it can be used mainly for obtaining performance parameters. The parameters that may be of interest are Mean_Burst_Size, Mean_Idle_Time, System_Throughput, Mean_System_Delay, and System_Utilization.

In order to obtain throughput and utilization, the related formulation is given below:

$$\text{Throughput} = \frac{\text{Number\_of\_Burst\_Sent}}{\text{Test\_Time}}, \tag{1}$$

$$\text{Utilization} = \frac{\text{Test\_Burst\_Time}}{\text{Test\_Time}}. \tag{2}$$

Therefore, we need to use a counter to count the number of bursts sent and two variables to keep the timing values for burst times and test duration. These variables

can also be used to calculate Mean_Burst_Size and Mean_Idle_Time according to the formulation given below:

$$\text{Mean\_Burst\_Size} = \frac{\text{Total\_Burst\_Size}}{\text{Number\_of\_Burst\_Sent}}, \tag{3}$$

$$\text{Mean\_Idle\_Time} = \frac{\text{Test\_Time} - \text{Total\_Burst\_Time}}{\text{Test\_Time}}. \tag{4}$$

In order to obtain the last parameter, System Delay, we need three more timing variables. The first one will be used to keep tract of the time at the source node when the Setup message is sent to the Ingress switch. The second one keeps the time value at the destination when the connection is closed due to a Release message received or a forced release by a timeout message. The last one is used to store the difference of these two variables to calculate total time spend for transmission. Once we computed that total time, the mean system delay is given

$$\text{Mean\_System\_Delay} = \frac{\text{Total\_System\_Delay}}{\text{Number\_of\_Burst\_Sent}}. \tag{5}$$

Once these parameters are obtained, they can be used as inputs to analytical analysis.

4. Conclusions

In this paper, we present a formal description of the jumpstart just-in-time signaling protocol for unicast traffic. We define single burst unicast signal flows. In the single burst unicast connection, we set a connection only for sending one burst and then close the connection. The state diagrams and transitions are given in this paper. Once the state diagrams are given, we also obtain the reachability trees for each entity and basing on these trees we create UIO sequences. The future work includes generating testing sequences for persistent type connections and the definition of multicast traffic which is part of the jumpstart just-in-time protocol specifications.

Acknowledgements – This research effort is being supported through a contract with ITIC (Intelligence Technology Innovation Center).

References

[1] NAIL K., SARIKAYA B., *Testing communication protocols*, IEEE Software **9**(1), 1992, pp. 27–37.

[2] ATES A.F., SARIKAYA B., *Test sequence generation and timed testing*, Computer Networks and ISDN Systems **29**(1), 1996, pp. 107–31.

[3] LAI R., *A survey of communication protocol testing*, Journal of Systems and Software **62**(1), 2002, pp. 21–46.

[4] SIDHU D.P., LEUNG T.-K., *Formal methods for protocol testing: a detailed study*, IEEE Transactions on Software Engineering **15**(4), 1989, pp. 413–26.
[5] KING P.W., *Formalization of protocol engineering concepts*, IEEE Transactions on Computers **40**(4), 1991, pp. 387–403.
[6] HANSSON H., JONSSON B., ORAVA F., PEHRSON B., *Formal design of communication protocols*, International Switching Symposium 1990, pp. 99–104.
[7] HOLZMANN G.J., *Protocol design: redefining the state of the art*, IEEE Software **9**(1), 1992, pp. 17–22.
[8] TURNER K.J., *The Use of Formal Methods in Communications Standards*, IEE Colloquium on "Formal Methods for Protocols", 1991, pp. 1/1–3.
[9] BALDINE I., ROUSKAS G.N., PERROS H.G., STEVENSON D., *Jumpstart: a just-in-time signaling architecture for WDM burst-switched networks*, IEEE Communications Magazine **40**(2), 2002, pp. 82–9.
[10] WEI J.Y., MCFARLAND R.I., *Just-in-time Signaling for WDM Optical Burst Switching Networks*, Journal of Lightwave Technology **18**(12), 2000, pp. 2019–37.
[11] YOO M., QIAO C.,DIXIT S., *QoS performance of optical burst switching in IP-over-WDM networks*, IEEE Journal on Selected Areas in Communications **18**(10), 2000, pp. 2062–71.
[12] TURNER J.S., *Terabit burst switching*, Journal of High Speed Networks **8**(1), 1999, pp. 3–16.
[13] QIAO C., YOO M., *Optical burst switching (OBS) – a new paradigm for an Optical Internet*, Journal of High Speed Networks **8**(1), 1999, pp. 69–84.
[14] YOO M., QIAO C., *Just-enough-time (JET): A high speed protocol for bursty traffic in optical networks*, IEEE/LEOS LEOS Summer Topical Meeting 1997, pp. 26–27.
[15] BOWMAN H., BLAIR G.S., BLAIR L., CHETWYND A.G., *Formal description of distributed multimedia systems: an assessment of potential techniques*, Computer Communicationsv **18**(12), 1995, pp. 964–77.
[16] ZAIM AH., BALDINE I., CASSADA M., ROUSKAS GN., PERROS, HG., STEVENSON D., *Jumpstart just in time signaling protocol: a formal description using EFSM*, Optical Engineering 42(2), 2003, pp. 568–85,

Received May 31, 2004