Volume 6, Issue 12, December 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Supporting Unordered Transport Delivery in Network Simulator

Amit Tripathi, Akhilesh Kosta
Kanpur Institute of Technology, Kanpur,
Uttar Pradesh, India

Abstract— With the increased use of internet and the rapid development of the applications running over it, the importance of TCP towards the applications has also changed. The signi cance of in-order delivery o ered by TCP has reduced with time since the required data framing/ordering strategies can now be easily implemented by the application APIs themselves. Also, the strict in-order delivery of TCP increases the average latency on the data segments which have to wait in the bu ers for the retransmission of an earlier transmitted data segment which was lost. The uTCP protocol o ered by the Minion architecture overcomes this drawback of TCP by providing a backward-compatible out-of-order delivery atop TCP with negligible CPU or bandwidth overheads. In this work, we have developed an implementation of uTCP as an extension for the widely used GNU Network Simulator 2. This extension supports out-of-order delivery similar to uTCP along with all the bene ts o ered by a legacy TCP connection like congestion control, connection state information, etc. The extension has been implemented as a derived class of the existing agent class Agent/TCP and has been developed in a manner that it can be easily extended as a derived class of other TCP variants like class Agent/TCP/Reno and class Agent/TCP/Vegas with minimal changes. Experimental results over the developed extension have con rmed that it delivers lower average latencies than TCP for multistreamed data transfers.

Keywords— TCP's Latency Overhead, UDP, Minion Architecture, u TCP- Unordered TCP, uCOBS Operation ns - The Network Simulator, Modi cations.

I. INTRODUCTION

Due to The reliable byte delivery o ered by TCP [6] [18] is certainly the backbone of present day internet. A large number of applications have been bene ting from this end-to-end communication service. The sender application generates data and simply writes it to the socket and the receiver reads the data from the socket. All the complexities involved in the successful transmission of data through the network, including an assurance that every single byte reaches the other end even if a packet gets dropped en route, are invisible to the application. TCP has thus ordered applications a convenient, reliable and high-level communication abstraction with semantics that are similar to the Unix le I/O or pipes to transfer data over the internet. The convenience has been further enhanced with the Unix tradition of implementing TCP in the OS kernel, allowing the application code to ignore the difference between an open disk le, an intra-host pipe, or an inter-host TCP socket. However, no new transport protocol has seen a large-scale deployment since TCP Applications, which do not use TCP, rarely utilize new out-of-order transports such as SCTP [19] and DCCP [14], and rely on UDP[17] encapsulation to traverse through middle boxes. UDP has been a popular transport medium, but it is not universally supported over the internet due to reasons discussed in Section 1.1.2. This has led even delay sensitive applications like Skype [4] and Microsoft's Direct Access VPN to deploy their services over TCP inspit of its latency overhead.

II. METHODOLOGY

- Network simulators provide a cost effective method for
- Network design validation for enterprises / data centers /sensor networks etc.
- Analyzing Utilities distribution communication, railway signaling / communication etc
- Network protocol R & D
- Defense applications such as HF / UHF / VHF MANET networks, Tactical data links etc.
- There are a wide variety of network simulators, ranging from the very simple to the very complex. Minimally, a network simulator must enable a user to
- Model the network topology specifying the nodes on the network and the links between those nodes.
- Model the application flow (traffic) between the nodes.
- Providing network performance metrics as output.
- Visualization of the packet flow.
- Technology / protocol evaluation and device designs.
- Logging of packet / events for drill down analyses / debugging.

© 2016, IJARCSSE All Rights Reserved

2.1 Simulation Workflow of NS:

The general process of creating a simulation can be divided into several steps:

Topology definition: To ease the creation of basic facilities and define their interrelationships, ns-3 has a system of containers and helpers that facilitates this process.

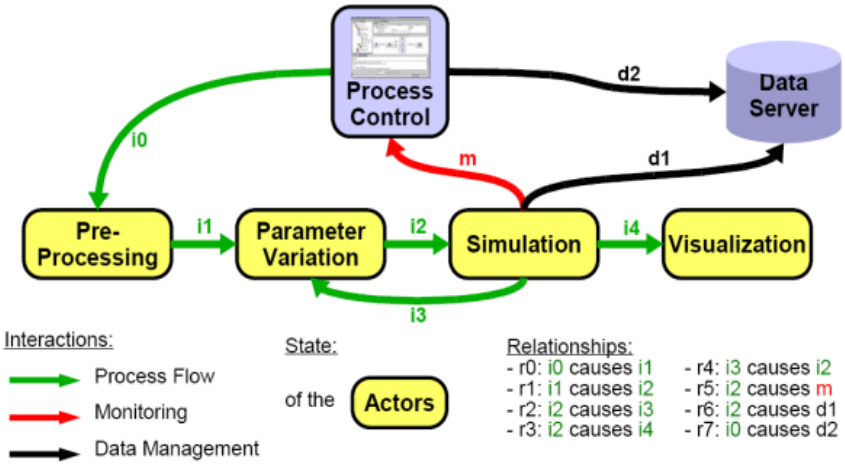
Model development: Models are added to simulation (for example, UDP, IPv4, point-to-point devices and links, applications); most of the time this is done using helpers.

Node and link configuration: models set their default values (for example, the size of packets sent by an application or MTU of a point-to-point link); most of the time this is done using the attribute system.

Execution: Simulation facilities generate events, data requested by the user is logged.

Performance analysis: After the simulation is finished and data is available as a time-stamped event trace. This data can then be statistically analysed with tools like R to draw conclusions.

Graphical Visualization: Raw or processed data collected in a simulation can be graphed using tools like Gnuplot, matplotlib or XGRAPH.

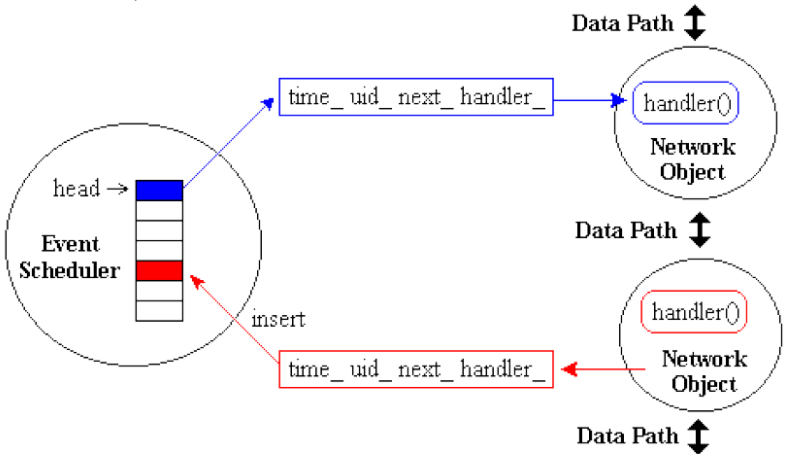


2.1.1 Event Scheduler

The event scheduler is responsible for keeping track of time during a simulation and ring the events in order from the event queue. The ring is done by invoking the handler of the appropriate network component. The event scheduler is critical to the functioning of the components which simulate the delay in packet-handling and the ones who need timers. In majority of the cases, an event is handled by the same object which issued that event. Figure 2.8 shows the outline of a network object using an event scheduler.Packets in ns follow two types of paths: data path and event path. Data path refers to the actual network path in the de ned network topology. Event path refers to the ow of control to handle the packets between the various layers (i.e. associated protocol agents) at a node or a link. Packet handling between two network objects comprises of two methods being invoked:

send(P acket p)ftarget!recv(p)g method of the sender

recv(P acket ; Handler h = 0) method of the receiver



2.2 Implementing a new protocol

The development of a new protocol in ns is done by implementing a new Agent class. Similarly, a new application is written under the Application class. This section gives an insight as to how a fully functional simulation, with a new application running over a new protocol, works in ns. A fully functional simulation which transfers data over uTCP in ns comprises of the following procedures in its ow of control:

Sender Application:

The rst trigger of control is from the simulation scenario written in a .tcl le where a line like

© 2016, IJARCSSE All Rights Reserved

$ns at 0:1 \$mmapp s start"
invokes the start() method of the application at the speci ed time. The application subsequently stores the data to be transferred at a particular memory location and invokes a routine of the underlying transport agent. An instance of this can be agent !sendmsg(msg; len) where msg and len are the location and number of bytes of the application data to be sent.

Sender Agent:

Once the send routine of the transport agent is invoked, the required processing and data handling is done. For uTCP, this includes en-coding the application data using uCOBS. Finally a packet is created and its header values are set. This also includes storing the permissible chunk of the application data in the packet. The packet is nally sent to the underlying network agent by invoking
target !recv(p; h)

where p and h are the pointers to the packet and an array of header ags. The above routine call is a standard call for invoking the recv(Packet* pkt, Handler*) method of the next agent. Since we are limiting our work to trans-port layer modifications, any further ow control can be left for ns.

Receiver Agent:

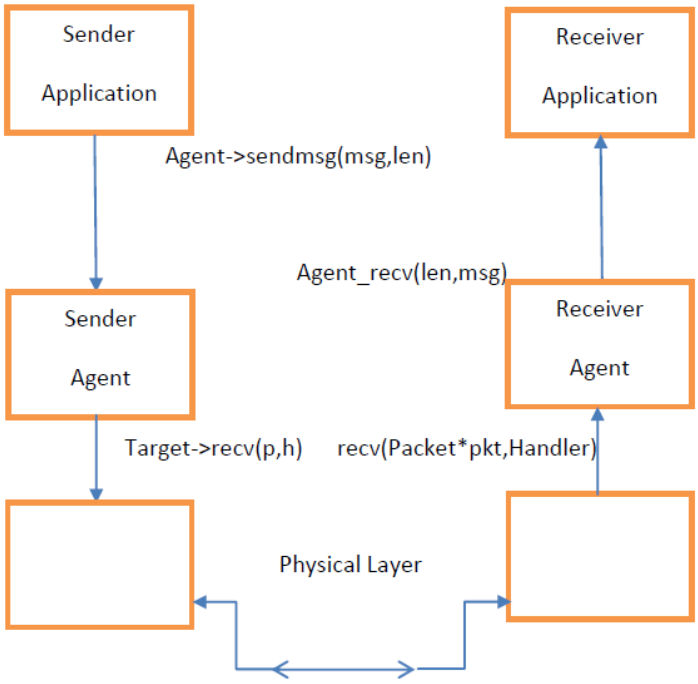
After sending the packet to the network agent at the source, the packet transfer is simulated over the de ned network topology. Eventually, the network agent at the destination invokes the recv(Packet* pkt, Handler*) method of the transport agent. This is followed by the required packet handling and data processing.For uTCP, this step includes storing the packet data in a linked list of byteblock structures, followed by an attempt to extract a complete message from the queue. If both the zero marker bytes of a message are not found, we move further in the queue to extract other messages if complete. This searching in the queue accounts for the out-of-order delivery of messages which is not permitted in the TCP bu er due to the semantics of TCP.Finally, the data is passed on to the receiver application using the routine call
app !recv msg(len; msg)

The above call invokes the recv msg(int len, char * msg) method of the application with the pointer and length of the application data as len and msg respectively.

Receiver Application:

The receiver application receives the data through its recv msg method for further:

Table 4.1 Flow control of a ns simulation



2.3 Typical Modifications

We include three new attributes to the common header struct hdr cmn present in /n-2.35/common/packet.h.

char payload[65536]:

Memory to store the data generated by the applica-tion. The value 65536 can be changed as per requirements.

int data set:

This ag set is set whenever the payload eld of the packet is carrying application data

uint32 t tcp seq:

Analogous to the sequence number in a standard linux TCP header (bytes)All the new/modi ed public functions developed as part of a derived class of class Agent need to be declared duly in the /ns-2.35/common/agent.h and /ns-2.35/common/agent.cc in order to make them accessible to the application(s). Sim-ilarly, all the new public functions

© 2016, IJARCSSE All Rights Reserved

developed as part of any application need to be declared in the /ns-2.35/apps/app.cc and /ns-2.35/apps/app.h to make the accessible to the agent(s).

The Makefile present in /ns-2.35/ is updated to include the new object les, namely tcp/tcp-utcp.o, tcp/tcp-utcpsink.o, userspace.o, void.o, and mm-multi.

2.4 uTCP Sender Side Agent

For the purpose of implementing a uTCP sender, we develop a new class as a derived class of class Agent/TCP, namely class Agent/TCP/UTCP. The name of the corresponding agent is UTCPAgent and the corresponding C++ source and header les are /ns-2.35/tcp/tcp-utcp.cc and /ns-2.35/tcp/tcp-utcp.h . The various attributes of this class are -

int max data amount:

Maximum theoretical limit of the data (in bytes) allowed to be transmitted over a TCP connection

char* data bu er:

The position where the sender side TCP bu er is allocated (typically of the size of max data amount)

int total data:

Total amount of data (in bytes) transmitted till this instance

int map:**

A two-dimensional array which keeps a mapping of the seqno of transmitted packets to the position in the data buffer

char* read to:

Pointer to the location in the data buffer to where the next application data should be written to

char* to send:

Pointer to the location in the data buffer from where the next packet should read its data from

int len:

Number of remaining bytes of the application data to be transmitted

uint32 t tcp seq:

Analogous to the seq. number in a standard linux TCP header (bytes)

int done with data:

This ag is set by the sender when the application has no bytes to send and the uTCP agent has no outstanding data in the data buffer to be sent

int max seqno:

The maximum seqno sent till now by the uTCP agent The newer and modi ed functions of this class are -

void sendmsg2(char *data, int nbytes):

A modi ed implementation of sendmsg function of class Agent/TCP, this function is invoked by the application where the data pointer refers to the memory location having nbytes of application data. The Pseudo Code is described in Algorithm 1 Although ns does not perform any packet fragmentation currently, which is evident from the TCP acknowledgements being handled over seqno values from the packets, removing any requirement of COBS-encoding the data, we are still performing the encoding to ensure that the functioning of uTCP is independent of any future developments on routing mechanisms of ns.

Algorithm 1 void sendmsg2(char *data, int nbytes)

```
sizeOfCodedMsg cobs encodemsg(read_to,data,nbytes)
Update read to;
Update total_data;
Update_len;
Update_curseq;
Invoke send_much
```

int already delivered(int seqno):

This function simply checks, using the map, that if a packet with this particular seqno was sent earlier or not.

void output(int seqno, int reason):

This function copies the data from the data buffer to the packet payload and sets the data set ag. If the send request is not a retransmission, the length of data is taken as the minimum of amount of data to be transmitted and the packet size . The Pseudo Code of this function is as follows -

Algorithm 2 void output(int seqno, int reason)

```
if already delivered(seqno) then
Update this packet data from map;
Update packet data len from map;
Update this packet seq from map;
Update this packet data;
Update packet data len;
Update this packet seq;
Update map, map index;
Update to send, len;
```

© 2016, IJARCSSE All Rights Reserved

end if
/* Standard implementation of output routine of Agent/TCP continues */
Update size ;
Update is last;
Send the packet;

The new method, void sendmsg2(char *data, int nbytes) needs to be de-clared in the /ns-2.35/common/agent.h and /ns-2.35/common/agent.cc les in order to make it accessible by the application.

2.5 uTCP Receiver Side Agent

In order to accept out-of-order delivery of TCP packets at the receiver side, we implement a new class as a derived class of class Agent/TCPSink, namely class Agent/TCPSink/UTCPSink. The name of the corresponding agent is UTCPSink and the corresponding C++ source and header les are /ns-2.35/tcp/tcp-utcpsink.cc and /ns-2.35/tcp/tcp-utcpsink.h. The various attributes of this class are – int max data amount: maximum theoretical limit of the data (in bytes) allowed to be transmitted over a TCP connection struct blocklist nal data: this structure comprises of the start and end nodes of a doubly linked list which stores the data of every packet received; every node of this linked list is of type struct byteblock; the detailed de nition both the structures is described in Algorithms 3 and 4

Algorithm 3 Structure of blocklist

struct blocklist f
struct byteblock* head;
struct byteblock* tail;
g;

struct blocklist* queue:

Pointer to the final data blocklist; any new mes-sage is added to this queue when received

unsigned int cumulative seqnum:

The last sequence (bytes) number up till which the application data is continuous

int unorderedOptionOn:

which switches out-of-order delivery.The modified and new functions developed under this class are –

Algorithm 4 Structure of byteblock

struct byteblock f
struct byteblock* next;
struct byteblock* prev;
uint32 t seq;
unsigned char* data;
int size;
/* ag to check if the block has been delivered */ int delivered;
/* ag to check if subsequent block pointed by next is contiguous in sequence number */ int contiguous next;
g;

void recv(Packet* pkt, Handler*):

This function is invoked automatically whenever a new packet is received; if the data set ag is set, this method invokes the minion update function over packet.

void minion update(Packet* pkt, char * recv data):

This function adds the packet data to the blocklist queue and delivers any sensible data to the application. The Pseudo Code is as follows -

Algorithm 5 minion update(Packet* pkt, char *recv data)

M packet size;
if M > 0 then
handle duplicate and addToQueue;
end if
while 1 do
Bcobs recvmsg;
if B > 0 then
free recv data;
break;
end if
app !recv msg;
end while

int handle duplicate and addToQueue(unsigned char* in bu , int size in bu , uint32 t seq, struct blocklist *queue):

This function takes the data from in buff and adds it as a new byteblock to the queue. The Pseudo Code of this function is

© 2016, IJARCSSE All Rights Reserved

Algorithm 6 int handle duplicate and addToQueue(...)

new byteblock N;
Update byteblock N;
if (queue is null) then
cumulative seqno 0;
else
cumulative seqno last contiguous byte;
end if
Data of N in bu ;
INSERT N into Queue

cobs recvmsg(struct blocklist *queue, unsigned char* message, int maxlen):
This method checks if there is any message in the queue. If it does,
it returns the message. To be a message,:
{
it must be between two 0's
{
there should be no gap/hole in the sequence number space the implementation of this function includes walking through the queue along with COBS decoding.
}

III. CONCLUSIONS

Simulation in network research plays the valuable role of providing an environment in which to develop and test new network technologies without the high cost and complexity of constructing testbeds. While not a complete replacement for testbeds, a standard frame- work for simulation used by a diverse set of researchers increases the reliability and acceptance of simulation results. Despite the benefits of a common framework, the network research community has largely developed individual simulations targeted at specification studies due to the considerable e ort required to construct a general-purpose simulator. Because of the special purpose nature of such simulators, studies based on them often do not react the richness of experience derived from experimentation with a more extensive set of traffic sources, queuing techniques, and protocol models.

REFERENCES

[1] Effective isotropically-radiated power-wikipedia, the free encyclopedia. http://en.wikipedia.org/ wiki/Effective_isotropically-radiated_power..
[2] Media labs asia, iit madras. http://www.tenet.res.in.
[3] Soekris Engineering. http://www.soekris.com.
[4] Srtm repository. ftp://e0srp01u.ecs.nasa.gov.
[5] Madhuresh Agrawal. Optimum tower height assignments for long distance 802.11 links. Technical report, cs397 Report,Indian Institue of Technology, Kanpur, 2006.
[6] Daniel Aguayo, John Bicket, Sanjit Biswas, Glenn Judd, and Robert Morris. Link-level Measurements from an 802.11b Mesh Network. In SIGCOMM, Aug 2004.
[7] Pravin Bhagwat, Bhaskaran Raman, and Dheeraj Sanghi. Turning 802.11 Inside-Out. In HotNets-II, Nov 2003.
[8] S.A. Borbash and E.H. Jennings. Distributed topology control algorithm for multihop wireless networks. In IJCNN, 2002.
[9] Eric Brewer, Michael Demmer, Bowei Du, Kevin Fall, Melissa Ho, Matthew Kam, Sergiu Nedevschi, Joyojeet Pal, Rabin Patra, and Sonesh Surana. The Case for Technology for Developing Regions. IEEE Computer, 38(6):25{38, June 2005.
[10] Zhuochuan Huang, Chien-Chung Shen, C. Srisathapornphat, and C.Jaikaeo. Topology control for ad hoc networks with directional antennas. In ICCN, 2002.
[11] Paul Ipe. Power allocation issues in a wireless mesh network. Technical report, BTech Project Re-port,Indian Institue of Technology, Kanpur, 2004.
[12] Ashok Jhunjhunwala and Sangamitra Ramchander. Role of wireless technologies in connecting rural india. In Indian Journal of Radio & Space Physics, pages 363{372, 2005.
[13] Bhaskaran Raman and Kameswari Chebrolu. Design and evaluation of a new mac protocol for long-distance 802.11 mesh networks. In MOBICOM, 2005.

© 2016, IJARCSSE All Rights Reserved