Simulating Network Link Compression in Loss-less Wireless Sensor Networks (WSNs) Environment

Rozita Teymourzadeh
Computer Science Department
University of San Francisco
San Francisco, USA
rteymourzadeh@usfca.edu

Ramona Carmen Stoica
Computer Science Department
University of San Francisco
San Francisco, USA
rstoica@usfca.edu

Robert Chang
Simulation and Evaluation
Waymo LLC
Mountain View, USA
bobbychang@waymo.com

Vahab Pournaghshband
Computer Science Department
University of San Francisco
San Francisco, USA
vahab.p@usfca.edu

***Abstract*—Communication among wireless sensor networks (WSNs) utilize high power consumption and therefore the significant efforts have been taken to minimize the usage for low-power wireless applications. With the advent of the network compression algorithm, loss-less compression wireless solution attracts significant attention due to its property of reducing resources to store and transmit data and perfectly reconstruct the original data from the compressed data in a loss-less environment.**

To allow network researchers to simulate currently deployed networks, Network Simulator 3 (NS-3) needs to incorporate as many standard networking technologies as possible. One such technology is lossless payload compression in wireless network environments such as WSNs. While compression has been employed in various network environments, unfortunately, there is not yet an implementation for NS-3, a powerful, extensible, and popular simulator used for network research. This paper presents a verification of loss payload compression for the NS-3 environment. We strongly believe that the proposed tool contributes significantly to grow wireless researches and projects by simulating, analyzing, testing, and validating the network behavior.

***Index Terms*—wireless sensor networks (WSNs), loss-less compression, compression, zLib**

I. Introduction

Wireless sensor network (WSN) is designed to handle a large number of sensor data in the industry and home automation fields and it finds a significant application in the industry such as sensor monitoring, health check monitoring, inventory, and location-based service monitoring, factory and home automation and etc [1], [2].

Since more and more industries adapt to the wireless sensor networks solution and generate WSNs nodes, loss-less data compression is an essential solution to efficiently use the energy and to speed up the channel air traffic (frequency 2.4 GHz). Historically, data compression [3] topic has been in the research and the industry field for the past decades and has been attracted significant attention due to its efficiency and capability in data transmission and storage. It has been widely used to minimize disk utilization, decrease bandwidth consumption on the networks, and reduces energy consumption in the hardware [4].

Furthermore, network researchers and developers require to design and implement the compression communication system with no direct comparison data unless if they implement all competing mechanisms within every simulator that is a costly and time-consuming operation. It requires creating a comprehensive and advanced networking simulation environment. This limitation often causes simulations constructed by different groups to lack standardization and reputability. Unfortunately, as of today, NS-3 does not support a native implementation of compression and decompression algorithm on its network library which causes a slow-down on wireless and network research projects [5]. Hence, in this research work, we replicate the prior research work [6] to validate and verify compression library functionality and to enrich the simulation framework, as a benefit to the others researchers to concentrate on their own research and make use of compression extended simulation library for their research purposes. To enrich the simulation library, we target NS-3 [7] simulation environment and concentrate on the data compression, since data compression shows a lot of interest among the wireless network researchers.

On the other hand, NS-3 software is an open-source discrete-event network simulator that targets primarily research and educational use. NS-3 is available for research and development and a suitable tool for developing and extending the library under the C++ programming language.

In the recent research work and in the field of the data transmission and the networking area, Shimamura et al. [7] proposed a compression strategy meant to reduce the packages lost between the network based on the hypothesis that the advanced relay nodes are located inside the network, and they had computational, storage and network resources nodes [9].

In 2013, Songbin Liu et al. proposed a versatile compression method for floating-point data. They claimed their design can achieve much higher compression ratios than existing general-purpose compression methods with promising throughput and it can support asymmetric decompression [8].

Later, in 2016 Shamieh et al. [9] introduced an adaptive and distributed compression/decompression scheme. The new adaptive scheme was needed because of the unexpected growth of the traffic which triggered a set of incentives/challenges such as high congestion in the networks, high loss rates of the packets, and latency. The previously mentioned challenges had a higher impact on the performance of the network and at the same time had a negative impact on Quality of Service (QoS). The author suggested an adaptive and distributed compression/decompression model for reducing the congestion in the network and at the same time support/maintain the QoS.

Dong Tian [10] found the geometry distortion of point cloud measurement is a challenging task and proposed using point to plane distances as a measure of geometric distortions on point cloud compression. The intrinsic resolution of the point clouds was proposed as a normalized model to convert the mean square error to PSNR numbers [10].

By observing high utilization of data compression and data communication [11], [7], [12] and understanding the necessity of the data compression performance technique [13], [14], in this research work, the effort is taken to replicate a network compression link and validate compression functionality with zLib compression library. The NS-3 [5] simulation environment was targeted for further development and embed simulation library.

We provided a simulation tool by developing a wireless packet compression model in the NS-3 tool for wireless applications. We utilized a compression detection mechanism presented in section II, that detects effective compression by sending two packet trains with high and low entropy data. Finally, we reported a preliminary design of the compression network (section III) and the evaluation of our work, consisting of several tests that validate the implementation. The simulation result was shown and discussed (section IV).

II. System Architecture

While simulating wireless network link compression is implemented to avoid major modifications to the existing NS-3 code base, several modifications are introduced to support the compression functionality. In total, our implementation compression simulation requires modification in data-gram protocol (UDP) mostly in the *ns-3-allinone/ns-3-dev/src/point-to-point/model* library and in the following models:

```
   .udp-client.cc
   .udp-server.cc
   .point-to-point-net-device.cc
   .point-to-point.cc (application)
```

The transmit starts by wrapping the packet in the *point-to-point-net-device* model and sending it through the channel. This model is called on the channel to inform the physical device that has virtually started sending the signals. An event is scheduled for the time at which the bits have been completely transmitted. Here, *PointToPointNetDevice::SetCompressProtocolNumber* set the compress protocol number and in the receive method, the protocol number is checked for the possible match in order to conduct compression. *AddHeader()* and *RemoveHeader()* helper functions assist to remove outer layer of the packet and access to the protocol function.

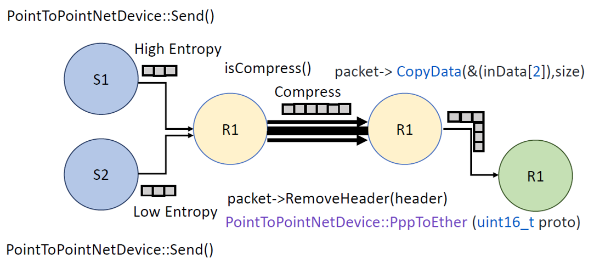

PointToPointNetDevice::Send()
High Entropy
S1
isCompress()
packet-> CopyData(&(inData[2]),size)
Compress
R1
R1
R1
S2
Low Entropy
packet->RemoveHeader(header)
PointToPointNetDevice::PppToEther (uint16_t proto)
PointToPointNetDevice::Send()

Fig. 1. Five nodes topology.

Figure 1 depicts the control flow for the compression algorithm in relation to other components in the NS-3 environment. The function in used and the helper functions are shown in the figure. *isCompress()* function get access to the protocol to send the control signal for compression. Based on the *isCompress()* return value, the packet is compressed using zLib library or is sent without alteration. In the other hand *PointToPointNetDevice::Receive (Ptr¡Packet¿ packet)* function controls the received packet to distinguish if the received packet is in compressed format. If the answer is *TRUE* the decompress process starts retrieving the compressed data in the R1 node shown in Figure 1.

Except for node S1 and S2 that always act as client and node R that acts as a server, the remaining nodes perform client and server and point to point node tasks. We have developed the 'application/model' directory of the NS-3 environment to simulate network link compression. Among the available models, we have implemented compression features in the UDP client and UDP server library model. In our implementation, the server echo feature is not supported, hence, the UDP server without the echo feature provides a suitable server environment for the development. Therefore, *udp-server.cc* and *udp-client.cc* were targeted to support compression. On the client-side, the Entropy attribute was implemented as shown here:

```
.AddAttribute ("Entropy",
  "Boolean Value",
  BooleanValue(true),
  MakeBooleanAccessor(&UdpClient::m_entropy),
  MakeBooleanChecker());
```

In addition, *std::vector <bool> generateRandomSequence()* function was implemented to support high entropy packet train and placed in the client model for testing purpose. Generate random sequence model utilized Linux utility to produce random sequences. Generated data were wrapped with the packet layers and several packet headers were added sequentially.

In order to send the packets via the network, *UdpClient::Send (void)* method was modified to provide support for compression link. On the other hand, udp-server.cc was responsible to handle receiving the packets and hence, *UdpServer::HandleRead* (*Ptr<Socket>socket)* was introduced to provide the read functionality. Time recording was handled in the *HandleRead()* function to generate the arrival time of the packet trains and generate a timestamp from local Linux time.

A list of the available application programming interface (API) is provided here. The introduced APIs allow developers to inherit the library and implement their applications on top of our implemented simulink models. Provided APIs help to enrich the NS-3 library for network simulation link compression.

```
bool isHighEntropy; /* entropy flag */
bool m_entropy; /* entropy flag */
virtual void SetEntropy(bool
    isHighEntropy); /* Set entropy flag */
void Send (void); /* Send the packet after
    packet preparation */
uint32_t m_count;/* Maximum number of
    packets the application will send */
Time m_interval; /* Packet inter-send time
    */
uint32_t m_size; /* Size of the sent
    packet */
uint32_t m_sent; /* Counter for sent
    packets */
uint64_t get_received (void) const; /*
    Receive the packet in server side */
void handle_read (Ptr<Socket> socket); /*
    Handle parsing packet after receiving
    */
uint64_t m_received; /* Number of received
    packets */
void Receive (Ptr<Packet> p); /* Receive
    packets */
virtual bool Send (Ptr<Packet> packet,
    const Address &dest, uint16_t
    protocolNumber);
/* Send packets */
bool doCompress = false; /* Compress flag
    */
bool doDecompress = false; /* Compress
    flag */
virtual void SetDecompressFlag (bool
    isRouter1); /* Set flag */
virtual void SetCompressFlag (bool
    isRouter1); /* Set flag */
int compressProtocolNumber=0;
virtual void SetCompressProtocolNumber
    (int protocolNumber); /* Set protocol
    */
```

Detail implementation is provided in the system verification.

III. System Design

Here, we provide a topology to verify and validate a compression functionality for model testing and system verification. To simulate network link compression, a network topology was implemented in the Point-to-Point Protocol (PPP, RFC 1661) [5] model library. The native PPP link is assumed to be established and authenticated at all the time.

At first, four nodes topology was designed that failed to send high entropy and low entropy packet trains concurrently for measurement purposes and automation. Hence, the five nodes topology with two senders was replaced with the original design.

Essentially, the topology of a communication compression network determines the path by which the packets travel from the source to the destination when compression links are enabled. connections between the communication nodes and the network are implemented in one batch, rather than in an incremental manner to validate the compression effect on the packet size. In addition, the proposed topology assumed that the physical infrastructure of the communication network already exists. In this implementation, the high-speed application becomes considerable especially when there is high traffic in the network. In order to generate high traffic, a single topology with five nodes was implemented using point to point protocol [6] standard library in the NS-3 environment [5].

Figure 2 shows the topology used in this project. The middle link was used to generate the bottleneck link. First, two nodes and the last node act as client and server applications where the senders send two sets of 6000 user UDP packets back-to-back as packet train, and the receiver records the arrival time between the first and last packet in the train. The first packet train consists of all of the size 1100 bytes in the payload, filled with all zeros, while the second packet train contains a random sequence of bits. This configuration generates high traffic in the middle point-to-point link that makes it a suitable environment to implement data compression on the top of the application.

In the subsequent project, a brief introduction to compression is provided as the main methodology to design a high-speed network. Then the smart algorithm for optimizing the topology is implemented. Finally, the statistical analysis is conducted to verify the functionality of the proposed design.

In order to extend NS-3 simulation library with the data compression feature, we need to implement and verify the compression functionality in the compression link topology. Furthermore, to fully utilize the network path for high-speed application, especially in the bottleneck of the network data compression is implemented and integrated.

Compressing network data is called deflating, and the algorithm is called DEFLATE. In this project zLib external library was implemented to compress data in the pre-link path. The zlib [16] is a software library used for data compression and is an abstraction of the DEFLATE compression algorithm used in their gzip file compression program. It is a loss-less data compression algorithm based on the principle of a dictionary-based encoding scheme.

The zlib library is also a crucial component of many software platforms as the compressed data format is itself portable across platforms. The side benefit of the compression algorithm is to reduce the memory footprint, speed up the packet transmission and regularization.

The compression link application was embedded and built using an NS-3 simulation environment with Lempel-Ziv-Stac

(LZS) [16] compression algorithm. It is a sliding-window compression algorithm and fixed Huffman coding and is responsible for compression and decompression of incoming and outgoing packets. The LZS algorithm follows RFC 1974 PPP Stac LZS Compression Protocol [16] requirement.

For the purposes of this project, the entire packets are inspected upon arrival for checking the protocol number. Matched protocol packet then is pushed for the compression process. The pre-processing finds the matched packet to replace the original protocol number with the LZS protocol number, here is 0x4021.

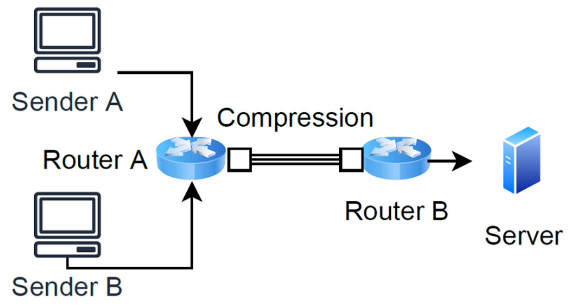


Fig. 2. Implemented 5 nodes topology with point-to-point compression link.

In the compressed structure, the original protocol number is appended to the original data, and compress that whole bit-string and replace it with the original data section in the original packet, as illustrated in Figure 2.

On the other hand, decompression, at the other side of the compression link then reverses all the pre-processing steps performed at the compression in the post-processing stage, to retrieve the original incoming packet, before pushing it to the next interface. The assumption is the two routers have already reached an Opened state and LZS [17] has been negotiated and removed the original header as the primary compression algorithm. Similarly, the decompression stage negotiates and appends the removed header in the compression stage to retrieve the original packet.

The compression and decompression algorithm is implemented in *PointToPointNetDevice* to enable compression and decompression on the point-to-point links in the NS-3 environment with the integrated LZS algorithm and supports full features of the zLib library.

The higher level of compression and decompression implementation model is shown in Figure 3.

IV. System Validation

To verify and validate the compression functionality, we replicated network compression detection introduced by Pournaghshband et al. [6] only in the cooperative environment as described. Therefore, our system validation topology, implementation details, and the parameters are completely based on the approach introduced in [6].

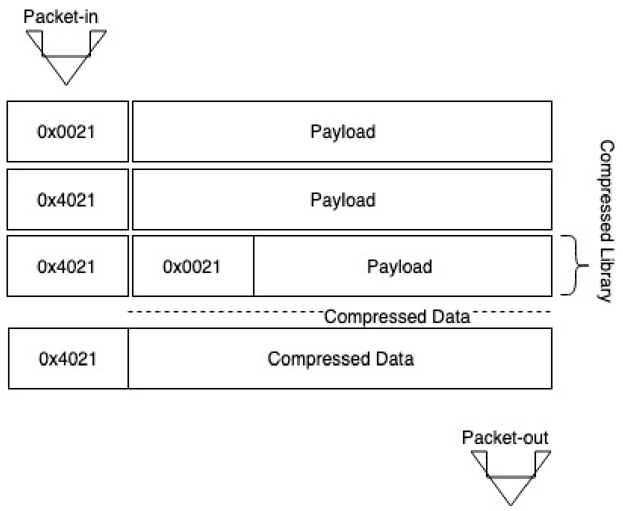


Fig. 3. The specified method to implement for assembling compressed packets, as specified in RFC 1974 [16].

The proposed network application is implemented as a client/server application where the sender sends two sets of 6000 UDP packets as a packet train one zero payload and the other random generated payload. The random sequence generator is implemented using /dev/random library in a Linux environment. In order to detect compression, a threshold value is introduced as described in prior research work for compression [6].

Although, there is no threshold for success for compression detection and the compression classification has never been attempted as a machine learning problem [18], [19] based on the prior research work criteria [6], we have selected 100 ms threshold for the compression detection. If the difference in arrival time between the first and last packets of the two trains is more than a fixed threshold of 100 ms, the application reports a compression detection event, whereas when the time difference is less than 100 there is probably no compression link on the path and the application reports no compression event is detected.

$$\Delta tH_{HighEntropy} = T_{FirstPktArrival} - T_{LastPktArrival}$$

$$\Delta tL_{LowEntropy} = T_{FirstPktArrival} - T_{LastPktArrival}$$

$$DetectionFactor = \Delta tH - \Delta tL \tag{1}$$

The compression algorithm is configured with one command-line argument, *Compression Link Capacity*, which specifies the maximum bandwidth across the link between the two routers. compression link capacity is tuned to find the compression efficiency. The implementation of the add-on feature in *SEND()* packet model in the NS-3 library is shown here:

```
/*
* Compression Model Implementation
*/
 if(this -> doCompress){
   PppHeader header;
   packet-> RemoveHeader(header);
   if(header.GetProtocol()==this->
   compressProtocolNumber){
   /* Remove IPV4 Header */
   Ipv4Header ipHeader;
   packet-> RemoveHeader(ipHeader);
   /* Remove UDP Header */
   UdpHeader udpHeader;
   packet-> RemoveHeader(udpHeader);
   /* Add Data & protocol and copy to inData
       */
   int protocolSize = 2;
   int dataSize = packet-> GetSize();
   uLongf size = dataSize + protocolSize;
   uint8_t *inData = new uint8_t[size];
   inData[0]=0x00;
   inData[1]=0x21;
   packet-> CopyData(&(inData[2]),size);
   uint8_t *compressData = new uint8_t[size];
   uLongf new_size;
   int returnValue =
       compress2((uint8_t*)compressData,
   &new_size, (uint8_t*)inData
       (uLongf)size,Z_BEST_COMPRESSION);
   size = new_size;
   packet = new Packet(compressData,size);
   size = size + BYTE;
   udpHeader.ForcePayloadSize(size);
   packet-> AddHeader(udpHeader);
   ipHeader.SetPayloadSize(size);
   packet-> AddHeader(ipHeader);
   header.SetProtocol
   (COMPRESSED_PROTOCOL_NUMBER);
   }
   packet->AddHeader(header);
 }
```

Fig. 4. Compression model implementation.

System testing and validation is conducted to ensure the correctness of the compression model library as an extended NS-3 library element.

As discussed earlier, the network topology was implemented in the point-to-point protocol library. The proposed project was run under a 5-nodes topology in an NS-3 environment as shown in Figure 2.

Nodes sender and receiver are the end-hosts running the network application. Nodes R1 and R2 are the intermediate routers where the link between them is *compression-enabled*. The project topology includes four logically separate simulations, each doing one of the following:

- Transmit low entropy data over a network topology without a compression link.
- Transmit high entropy data over a network topology without a compression link.
- Transmit low entropy data over a network topology with a compression link.
- Transmit high entropy data over a network topology with a compression link.

The control for confounding variables across the simulations was defined precisely, as it ultimately is comparing the time between the simulation types to detect the compression link. The whole system is automated to run the first low entropy packet train followed by the high entropy packet train.

The middle capacity link (router A, router B) then varies and tuned with different capacity parameters from 1 to 10 Mbps, reporting each ΔtH - ΔtL. The link capacity of the two routers (non-compression) links on the four-node topology were set persistently to 5 Mbps. The proposed two senders in the topology support low and high entropy respectively. Sender and receiver nodes are the UDP server and client. Receiver nodes record the arrival time between the first and last packets in the simulation. Four main components are implemented in the validation architecture.

- Configuration Management System
- Topology Management System
- Compression System
- Synchronization and Output Recorder

Protocol check, payload management, compression and decompression features were hosted in the *point-to-point-net-device* class library. The *UDP-client* model class was modified to generate a random payload in high entropy traffic and *UDP-server* model class hosts the entire calculation and compression detection algorithm.

The external *zLib* [16] - An abstract of the DEFLATE compression algorithm - embedded with the project to perform compression task and as such, the output data length of compression link altered due to compression process.

In order to build the project, *./waf* build system was implemented. the following commands built the project while the compression system was enabled.

```
/*
* To compile and build the project
*/
 $ cd workspace/Transport-Layer-Security
   /ns-3-allinone/ns-3-dev
 $ ./waf configure
 $ ./waf build
 $ ./waf --run "scratch/point2point
     --IsHighEntropy=1 --IsCompress=1
     --MaxPacketCount=2"
```

In the topology management, two UDP clients, a UDP server, and point-to-point net devices were initiated. The Initialized class implementation is shown for low entropy UDP clients. The device attribute for the link between router one and router two was set for data rate, delay and maximum transmit unit (MTU).

The compression and decompression flag were set while initializing the topology. The point-to-point net device library was developed to support the following features:

- Protocol Checker
- Payload Management
- Compression and Decompression

SEND() method in point-to-point net device hosts compression algorithm, while *RECEIVE()* method hosts decompression algorithm. The arriving point-to-point protocol packets were parsed to capture protocol for possible matched protocol numbers. The payload will be taken to compress based on zLib library if the checked packet type matches one in the configuration file. The protocol will be replaced with LZS protocol number to indicate data compression. The newly generated packet is sent to the decompression router through the link.

In the other hand, *RECEIVE()* function decompresses the packet payload data after removing all necessary header files.

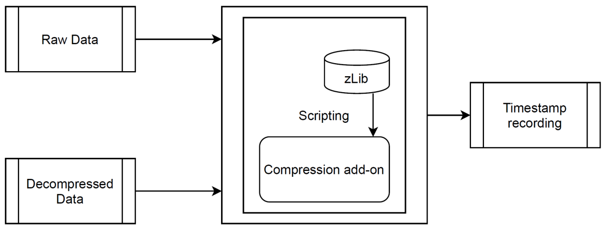


Fig. 5. Embedded add-on zlib library.

The implemented system is smart to detect and target the compressed packets for the decompression process. Figure 5 shows the involved header files with the arriving packets and e compression process on the packet. The second UDP client is then in charge to generate high entropy trained packet and run it through the network.

$$DetectionFactor = \Delta tH - \Delta tL \tag{2}$$

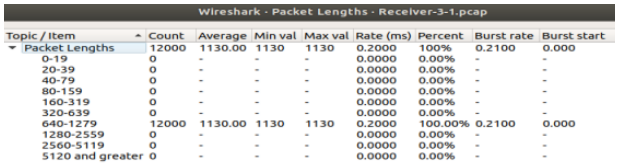
The same process takes place for high entropy trained packets. On the other hand, the UDP server counts the incoming packets and calculates the delta high entropy and the delta low entropy based on the equation 2.

To detect the compression, a threshold value of 100 ms is defined. The delta time difference more than the threshold value triggers a detection flag and vice versa. The timing report of the delta is logged and stored in the repository.

The verification process starts in the client and server application code. The sender nodes send 6000 UDP packet trains and the receiver records the arrival time between the first and the last packet in the trains. The first packet train consists of all packets of size 1100 (Figure 6) bytes in their payload and it is filled with zeros. On the other hand, the second packet train contains a random sequence of bits.

The arrival time of the first packet and the last packet (packet number 6000) were captured and plotted, whilst the capacity link data rate was increased from 1 Mbps to 10 Mbps.

Figure 7 shows the different timing between the first and the last packet when the compression is applied. It clearly shows


Wireshark · Packet Lengths · Receiver-3-1.pcap

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Packet Lengths	12000	1130.00	1130	1130	0.2000	100%	0.2100	0.000
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	0	-	-	-	0.0000	0.00%	-	-
80-159	0	-	-	-	0.0000	0.00%	-	-
160-319	0	-	-	-	0.0000	0.00%	-	-
320-639	0	-	-	-	0.0000	0.00%	-	-
640-1279	12000	1130.00	1130	1130	0.2000	100.00%	0.2100	0.000
1280-2559	0	-	-	-	0.0000	0.00%	-	-
2560-5119	0	-	-	-	0.0000	0.00%	-	-
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

Fig. 6. Successful arrival of all 12000 packets with the length of 1100 bytes.

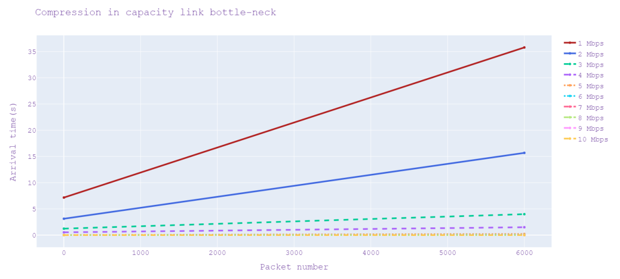


Fig. 7. Compression effect on the network in NS-3 library.

the compression is effective only and only if the bottleneck exists in the capacity compression link.

If the difference in arrival time between the first and the last packets of the two trains is more than the fixed threshold of 100 ms, the application reports *compression detected*, whilst when the time difference is less than the threshold value, there is probably no compression link in the path and the application reports *no compression is detected*. The middle link data rate was modified within the range of 1 Mbps up to 10 Mbps to generate a bottleneck of data on the link.

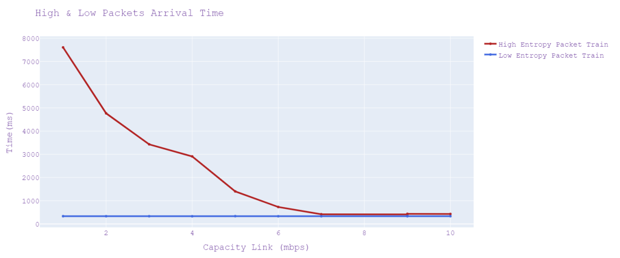


Fig. 8. High and low entropy train on the server node arrival time in NS-3 library.

Figure 7 shows the timing reported on the link capacity with low entropy packet train and high entropy packet train, while two outer links on the node topology were set to 8 Mbps consistently. The compression capacity link was altered to realize the compression behavior in different data rate configurations.

The time difference between high and low entropy packet trains is shown in Figure 8. While network bandwidth is steadily increasing, the arrival time of the low entropy packet train does not show much timing difference in compression link. On the other hand, there is significant behavior change is observed when high entropy packet train is located in the high bandwidth capacity link that proves the compression effect in the bottle-neck of the capacity link.

This effect has a high impact on wireless networks communication to utilize the wireless channels efficiently and allows high traffic wireless packets to travel in multiple wireless channels.

V. Acknowledgments

This research was supported by funding from Craig Newmark Philanthropies.

VI. Conclusions

Wireless sensor networks have reached about two billion in 2021 [20] worldwide and wireless efficiency and reliability is a challenging task among wireless researchers. This research has concentrated on extending the NS-3 simulation environment with a wireless data compression feature. We implemented and verified the data compression for WSNs application and we strongly believe our verified library extension benefits the wireless researchers and supports their project simulation.

We have replicated a prior research [6] embedded compression and fast decompression model to enhance the speed of data transferring in the bandwidth and to increase communication efficiency in the wireless network path. We have conducted testing and verification to validate the compression and decompression integrated model.

The external *zlib* library was embedded with the implemented topology to perform compression tasks in qualified wireless packets.

The two packet trains with low and high entropy were generated to be transferred via the compression capacity link for data evaluation and system verification. It is critical for analytic queries that repeatedly and synchronously read compressed packet data arrival time for analytic purposes and hence, The arrival time of the first packet train and the last packet train of each series were captured and plotted precisely. The result shows that compression uses the bandwidth efficiently when high load traffic is observed. However, the data compression is only effective when the capacity link turns to be a bottleneck.

Compression has no effect on the data transmission speed if there is no bottle-neck exists in the capacity compression link. The Compression and decompression model was implemented and validated for the NS-3 simulation library and the functionality was tested and verified for loss-less WSN applications.

References

[1] J. Gana Kolo, S. Anandan Shanmugam, D. Wee Gin Lim, L. Ang, K. Phooi Seng, "An Adaptive Lossless Data Compression Scheme for Wireless Sensor Networks", No. 37, Journal Sensors, Hindawi,Jul 2012, pp. 20.

[2] R. Teymourzadeh, A. Salah Addin, K. W. Chan, M. V. Hoong, "Smart gsm based home automation system", 2013 IEEE Conference on Systems, Process Control (ICSPC), 2013, IEEE, DOI: 10.1109/SPC. February 2013.6735152, pp.306–309.

[3] D. Lelewer, D. Hirschberg, "Data compression", ACM Computing Surveys (CSUR), 3rd ed., Vol. 19. ACM, September 1987, DOI: 296https://doi.org/10.1145/45072.45074, pp.261–296.

[4] B. Welton, D. Kimpe, J. Cope, C. Patrick, K. Iskra, R. Ross, "Improving i/o forwarding throughput with data compression", 2011 IEEE International Conference on Cluster Computing, September 2011, DOI: 10.1109/CLUSTER.2011.80 , pp. 438–445.

[5] U.S. National Science Foundation (NSF), "A discrete-event network simulator for internet systems, NS-3 Network Simulator", https://www.nsnam.org/, 2011-2021.

[6] V. Pournaghshband, A. Afanasyev, P. Reiher, "End-to-end detection of compression of traffic flows by intermediaries", 2014 IEEE Network Operations and Management Symposium (NOMS), May 2014, DOI: 10.1109/NOMS.2014.6838247 , pp.1–8.

[7] M. Shimamura,H. Koga, T. Ikenaga, M. Tsuru, "Compressing packets adaptively inside networks", IEICE transactions on communications, The Institute of Electronics, Information and Communication Engineers, July 2010, Vol. 93, No. 3, DOI: 10.1109/SAINT.2009.23, USA, pp.501–515.

[8] S. Liu, X. Huang, Y. Ni, H. Fu, G. Yang, "A versatile compression method for floating-point data stream", 2013 Fourth International Conference on Networking and Distributed Computing, December 2013, DOI: 10.1109/ICNDC.2013.32, IEEE, pp.141–145.

[9] F. Shamieh, A. Akhtar, X. Wang, "Adaptive distributed compression technique utilizing intermediate network nodes", 2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), ISBN: ISBN:978-1-4673-8722-4, May 2016, DOI: 10.1109/CCECE.2016.7726610 , IEEE, Canada, pp.1–4.

[10] D. Tian, H. Ochimizu, C. Feng, R. Cohen, A. Vetro, "Geometric distortion metrics for point cloud compression", 2017 IEEE International Conference on Image Processing (ICIP), September 2017, IEEE, DOI: 10.1109/ICIP.2017.8296925, pp.3460–3464.

[11] R. Serfozo, "Little Laws, Introduction to Stochastic Networks", Part of the Applications of Mathematics book series(SMAP), 1999, Vol. 44, Springer, pp.135–145.

[12] T. Ling Sun, L. Sei Ping, T. Chong Eng, "Enhanced compression scheme for high latency networks to improve quality of service of real-time applications", 8th Asia-Pacific Symposium on Information and Telecommunication Technologies, June 2010, ISBN: 978-4-88552-244-4, IEEE, pp.1–6.

[13] O. Shadura, B. Bockelman, "Root I/O compression algorithms and their performance impact within Run 3", Journal of Physics: Conference Series, Vol. 1525, No. 1, Apr 2020, IOP Publishing, DOI: 10.1088/1742-6596/1525/1/012049, Ser. 012049.

[14] A. Bulent, B. Bart, R. John, K. Matthias, M. Ashutosh, A. Craig , S. Bedri, B. Alper, J. Christian, S. William, M. Haren, W. Charlie, "Data Compression Accelerator on IBM POWER9 and z15 Processors : Industrial Product," 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), DOI: 10.1109/ISCA45697.2020.00012, May 2020, Spain, pp. 1–14.

[15] M.D. Schroeder, A.D. Birrell, M. Burrows, H. Murray, R.M. Needham, T.L. Rodeheffer, E.H. Satterthwaite, C.P. Thacker, "Autonet: a high-speed, self-configuring local area network using point-to-point links", IEEE Journal on Selected Areas in Communications, Vol. 9, Issue. 8, October 1991, DOI: 10.1109/49.105178, IEEE, pp.1318-1335.

[16] P. Deutsch, J. Gailly,"ZLIB Compressed Data Format Specification Version 3.3" 1996 RFC Editor, DOI: https://doi.org/10.17487/RFC1950, USA, 1996.

[17] A. Turpin, Y. Tsegay, D. Hawking, H. Williams,"Fast generation of result snippets in web search", Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR), DOI: 10.1145/1277741.1277766, July 2007, pp.127–134.

[18] D. Hahn, N. Apthorpe, N. Feamster,"Detecting Compressed Cleartext Traffic from Consumer Internet of Things Devices", Cornell University, DOI: 1805.02722v1 arxiv, May 2018, pp.1–7.

[19] J. Azar, A. AMakhoul, R. Couturier, J. Demerjian, "Robust IoT time series classification with data compression and deep learning", Neurocomputing, Vol. 398, 2011-2021, DOI: 10.1016/j.neucom.2020.02.097, 2020, Elsevier, pp. 222–234.

[20] P. Harrop, R. Das,"Wireless Sensor Networks 2011-2021", The new market for ubiquitous Sensor Networks (USN), IDTechEx report, www.IDTechEx.com/wsn, 2011-2021, pp. 340.