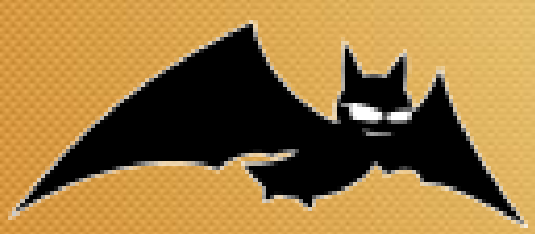
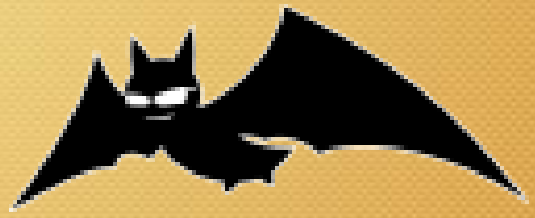
B.A.T.M.A.N-ADV AND BABLE HYBRID

- Shubham Aher
- Rishabh Pandita
- Swapnil Satpute





Problem Statement

BATMAN-ADV and BABEL Hybrid

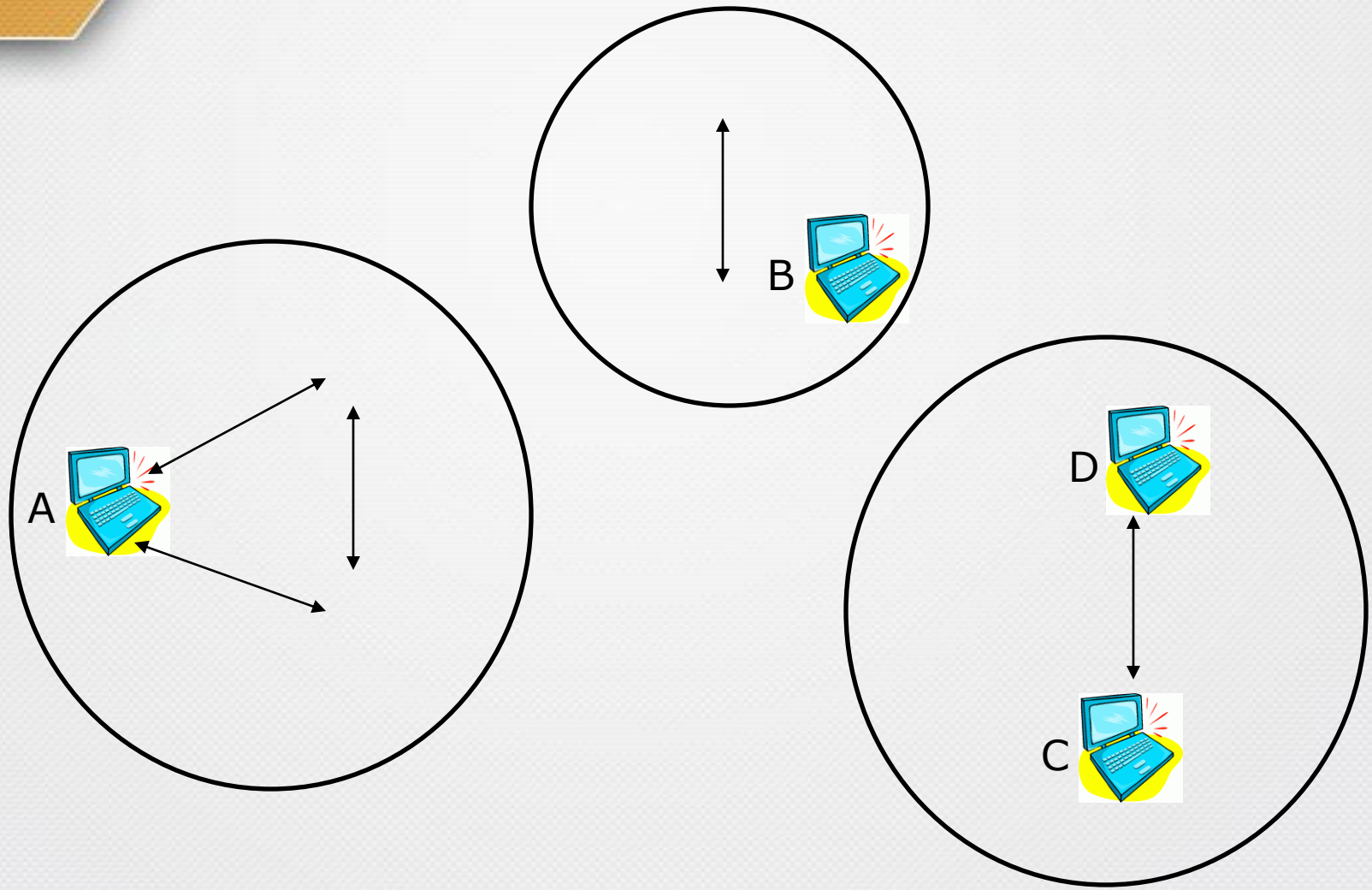
- To integrate B.A.T.M.A.N.-ADV and BABEL algorithms into a single routing algorithm and to carry out performance analysis for the same

What is MANET, BATMAN and BABEL ?

MANET
(Mobile Ad-hoc Network)

- Infrastructure based Networks vs. Ad-hoc
- Routing MANET
- Challenges for routing
  - Dynamic Topology
  - Device Discovery

Dynamic Topology in MANET



MANET Routing Approaches

- Better Approach To Mobile Ad-hoc Networking
  (B.A.T.M.A.N.-ADV)
- BABEL
- Others like OLSR, AODV, etc.

B.A.T.M.A.N-ADV

- Proactive
- Data Link Layer Protocol
- Network layer agnostic
- Nodes can participate in mesh without an IP
- Easy Integration of Non-Mesh clients
- Used in Linux kernel v2.6 and above

Working of B.A.T.M.A.N.-ADV

- Decentralized knowledge
- OGMs(Originator Messages) determine TQ(Transmission Quality)

BABEL

- Proactive
- Network Layer
- Distance vector routing
- User space daemon
- Loop free routing

Working of BABEL

- Neighbor cost based on ETX(Expected Transmission Count)
- Periodic broadcast of ‘Hello’ messages
- ‘IHU’ (I Heard You) messages as acknowledgments .
- History sensitive routing
- Feasibility check to avoid route loops.

Combining B.A.T.M.A.N.-ADV with BABEL

- Running babel daemon over B.A.T.M.A.N.-ADV directly without any modifications results into a highly unstable protocol
- Data bursts are very frequent, which are disastrous for wireless meshes
- Hence, modifications are required so that the protocols can function together as a single unit

Problem Statement Revisited

BATMAN-ADV and BABEL Hybrid

- To integrate B.A.T.M.A.N.-ADV and BABEL algorithms into a single routing algorithm and to carry out performance analysis for the same

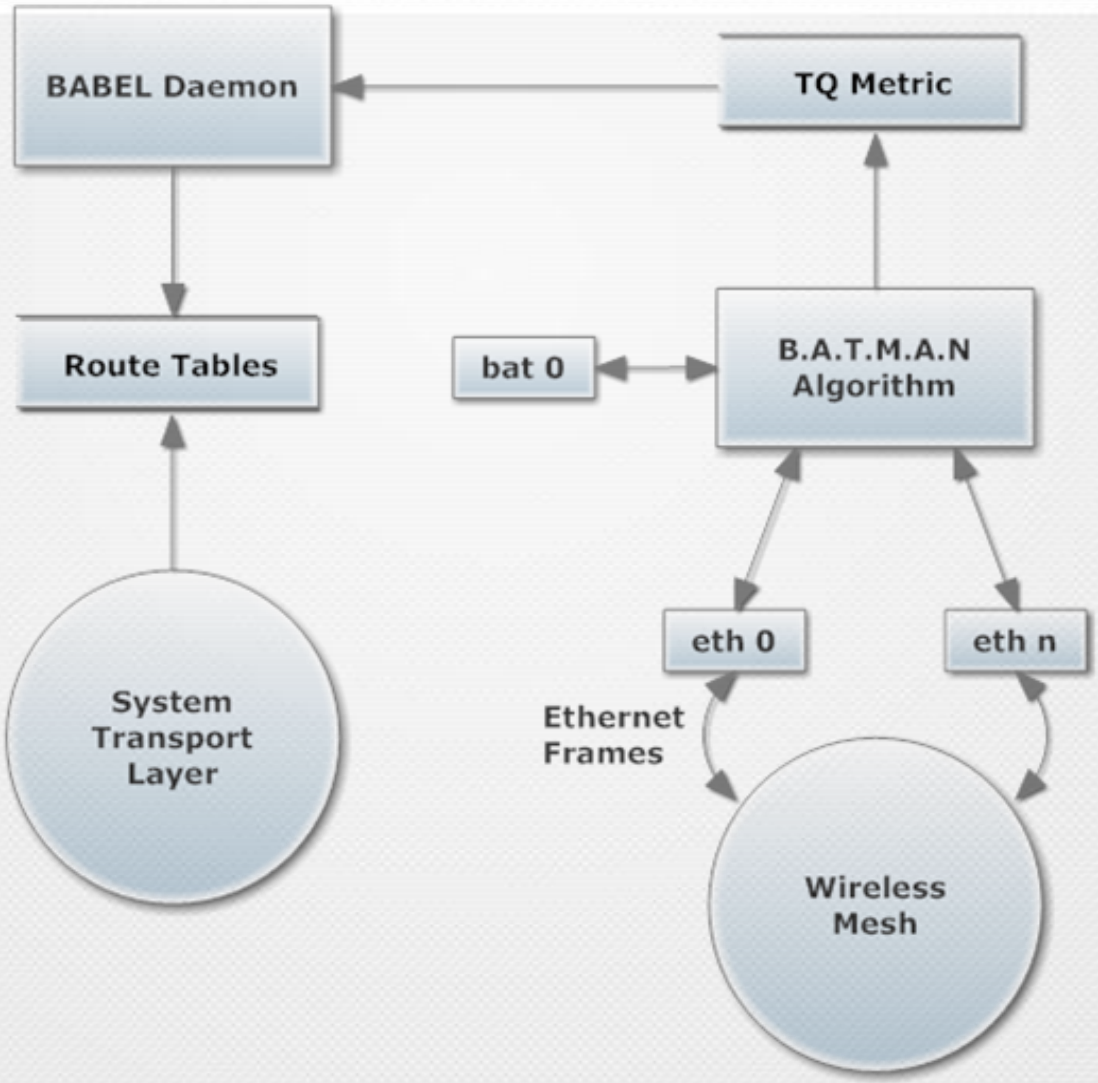
Why implement a hybrid protocol ?

- No protocol is perfect in general
  - Performance greatly depends on the nature of mesh (mobility, number of nodes, etc.)
- Proposed by Sven Eckelmann (OpenMesh community).
- It is an attempt to formulate a general solution for all kinds of meshes.

Behavioral Comparison of B.A.T.M.A.N.-ADV and BABEL

Factor	B.A.T.M.A.N.-ADV	BABEL
Implementation	Data link layer i.e. kernel space routing module	Network layer i.e. user space routing daemon
Throughput	Tends to route through more number of hops hence low multi-hop bandwidth	Higher multi-hop bandwidth
Packet delivery	Higher packet delivery rate	Lower packet delivery rate
Mobility	Slower route convergence and frequent route flapping is observed after repair.	Fast route convergence and repair
Scalability	Performance remains consistent as the no. of participating nodes rises.	Low round trip delay for few nodes but increases with the number of nodes
Stability	Higher stability	Route flapping may occur even for tiny changes

Hybrid Protocol Design



Metrics under consideration

- Babel metrics:
  - Tx Transmission Cost
  - Rx Reception Cost
  - ETX = 1/(Tx * Rx)
- B.A.T.M.A.N.-ADV metric:
  - TQ Transmission Quality (Bi-directional link quality)

Our Modifications

- During, route selection babel will select the neighbor with least cost as next hop
- Existing neighbor cost function in babel:

$$cost = ((Tx * Rx) + 128) >> 8$$

We know that ETX = 1/(Tx * Rx), also an increase in TQ value should decrease the neighbor cost

- Modified neighbor cost function in babel:

$$cost = (((Tx * Rx) + 128) >> 8) – TQ$$

Performance Measure

Protocols tested

- B.A.T.M.A.N – ADV and BABEL Without Any Metric Modification
- B.A.T.M.A.N – ADV and BABEL With Metric Modification
- B.A.T.M.A.N – ADV standalone
- BABEL standalone

Softwares Used for Benchmarking

- To Measure Bandwidth
  - Bwm-ng on Intermediate Nodes
- To Generate Traffic and measure performance
  - I-perf Client run on source node
  - I-perf Server run on destination node
- To Generate graphs
  - Gnuplot.py
  - J-perf to generate iperf data graphs
- Shell scripts to automate the testing

Test Bed

SOFTWARES USED

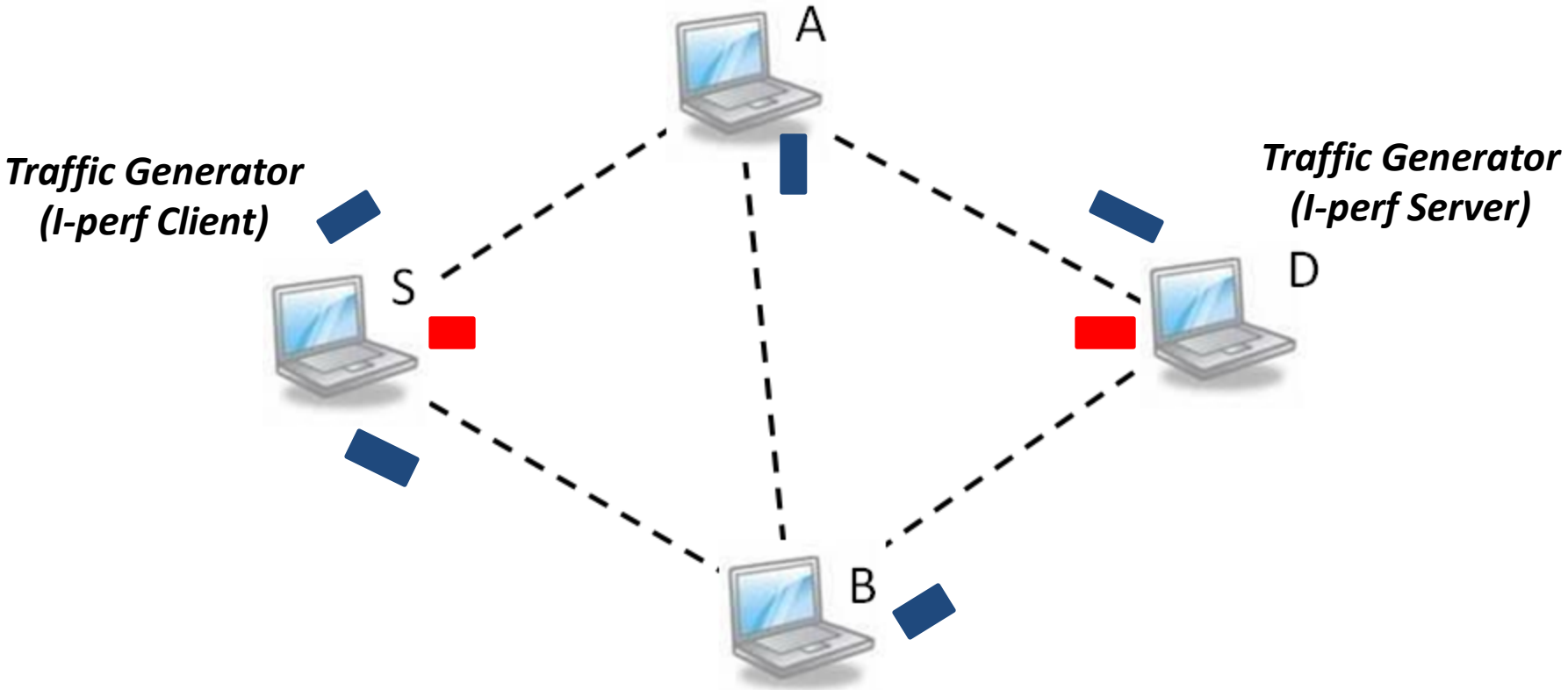
Network reachable

Unreachable network

A,B - Intermediate Nodes
S - Source Node
D - Destination Node

Bandwidth Monitor (bwm-ng)

S and D cannot reach each other

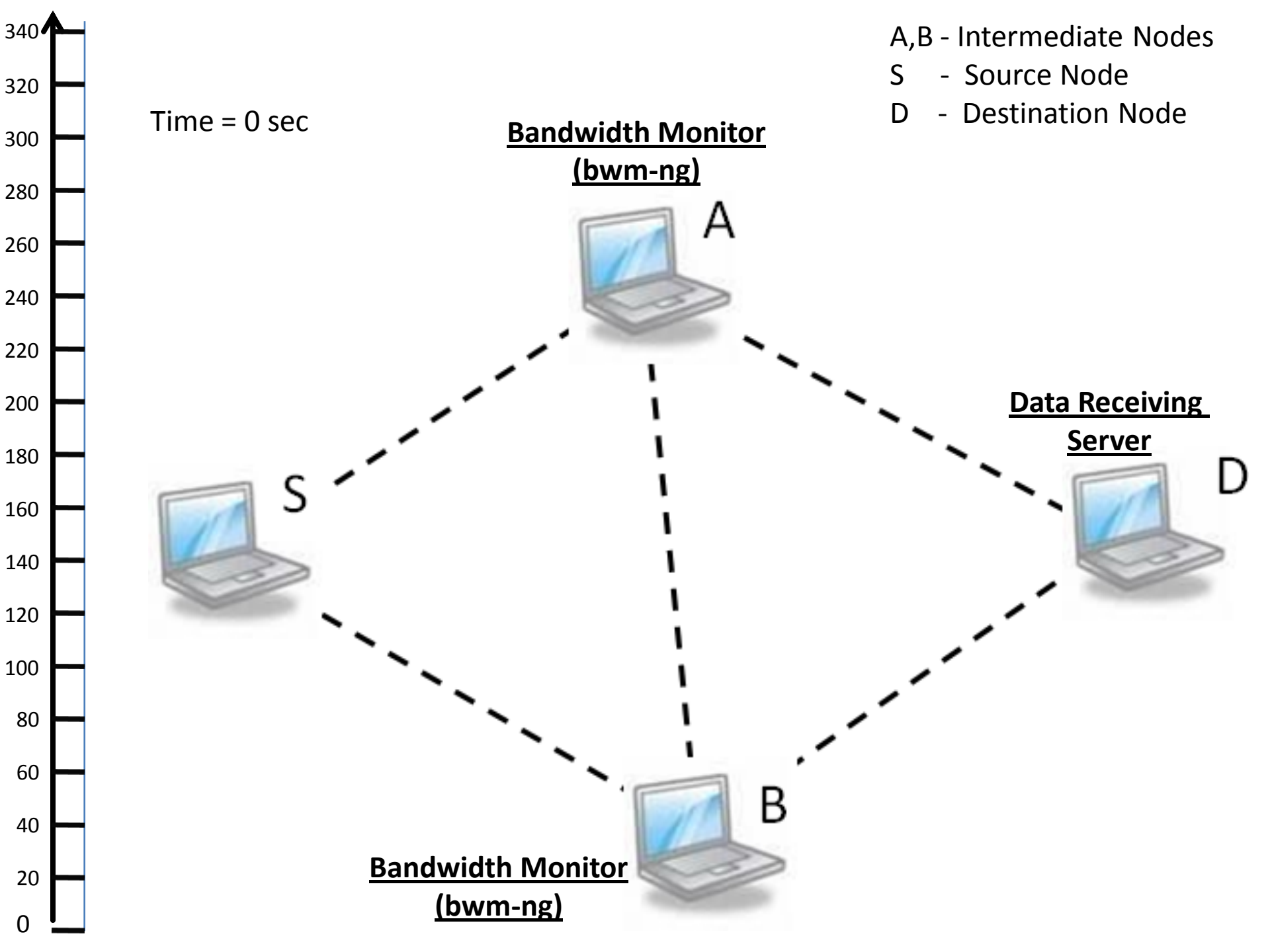


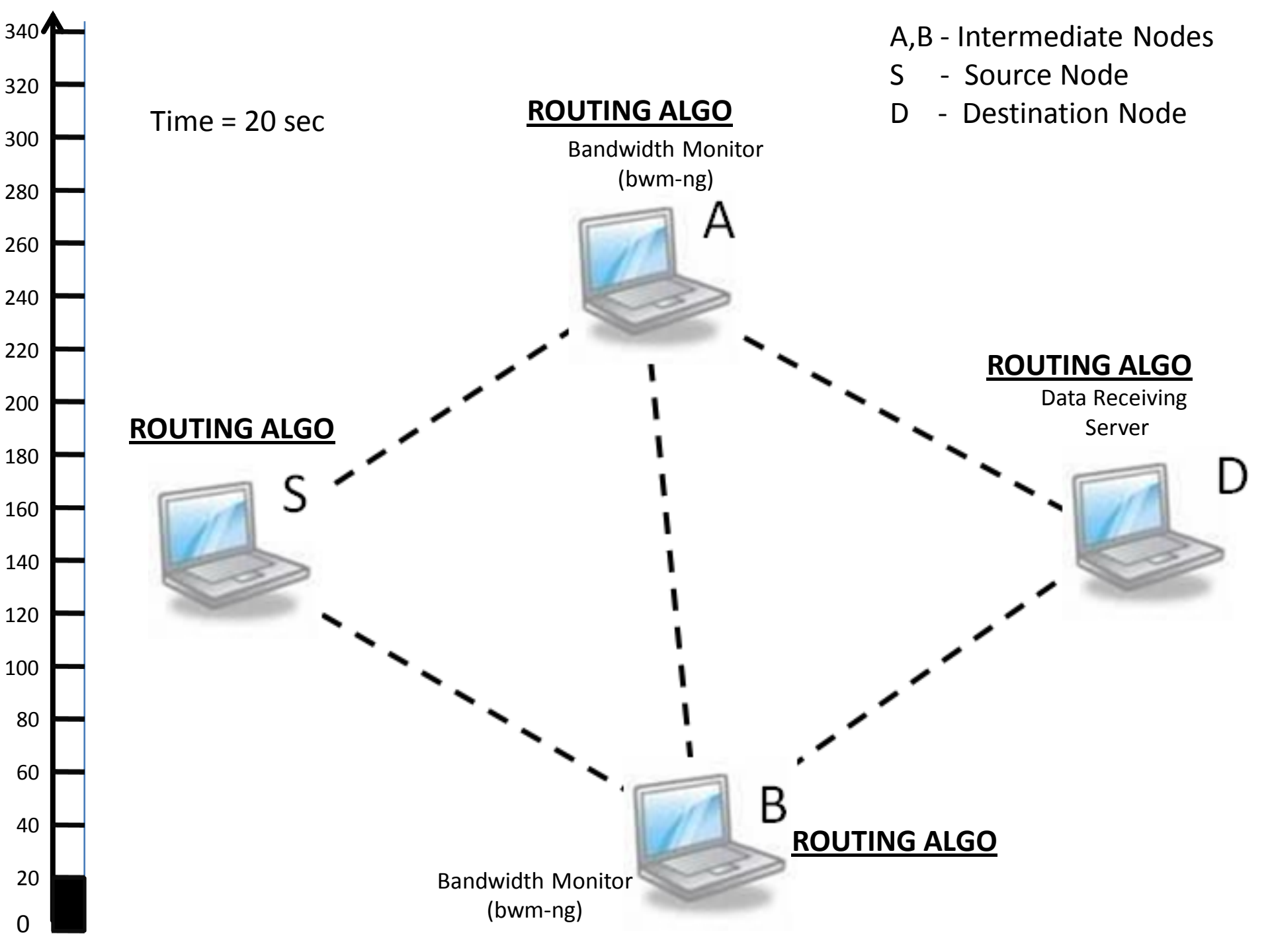
Traffic Generator (I-perf Client)

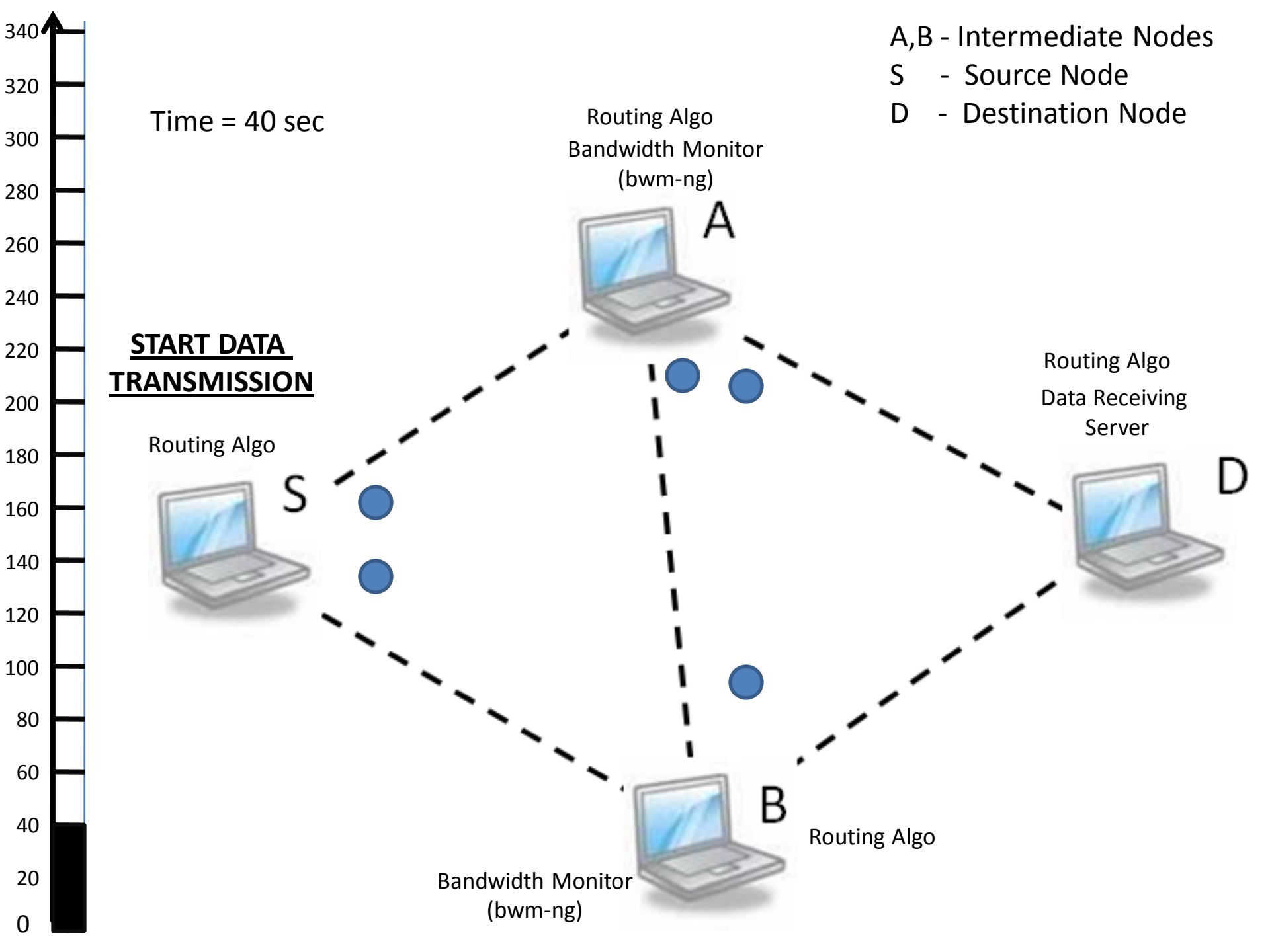
Traffic Generator (I-perf Server)

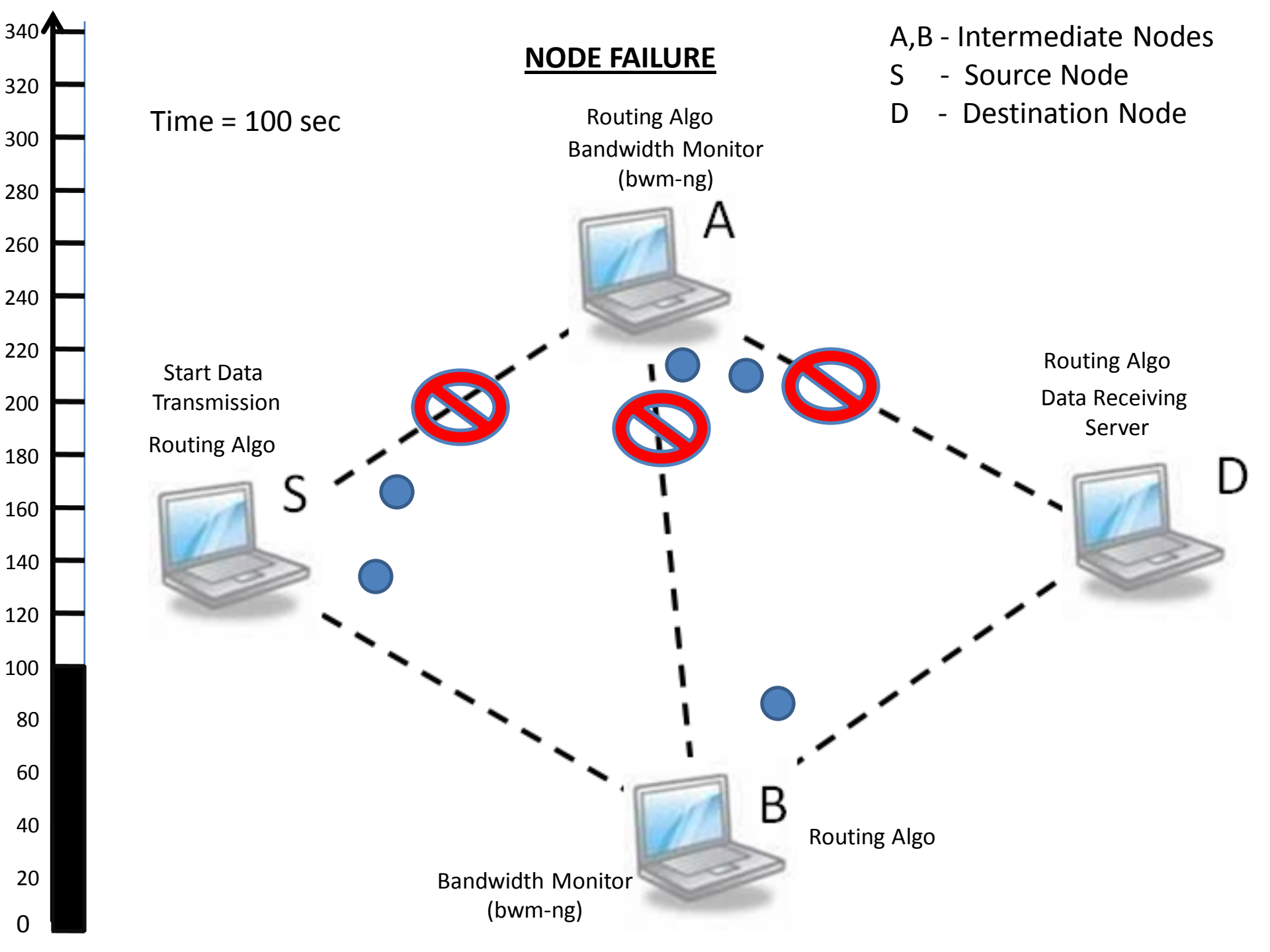
Bandwidth Monitor (bwm-ng)

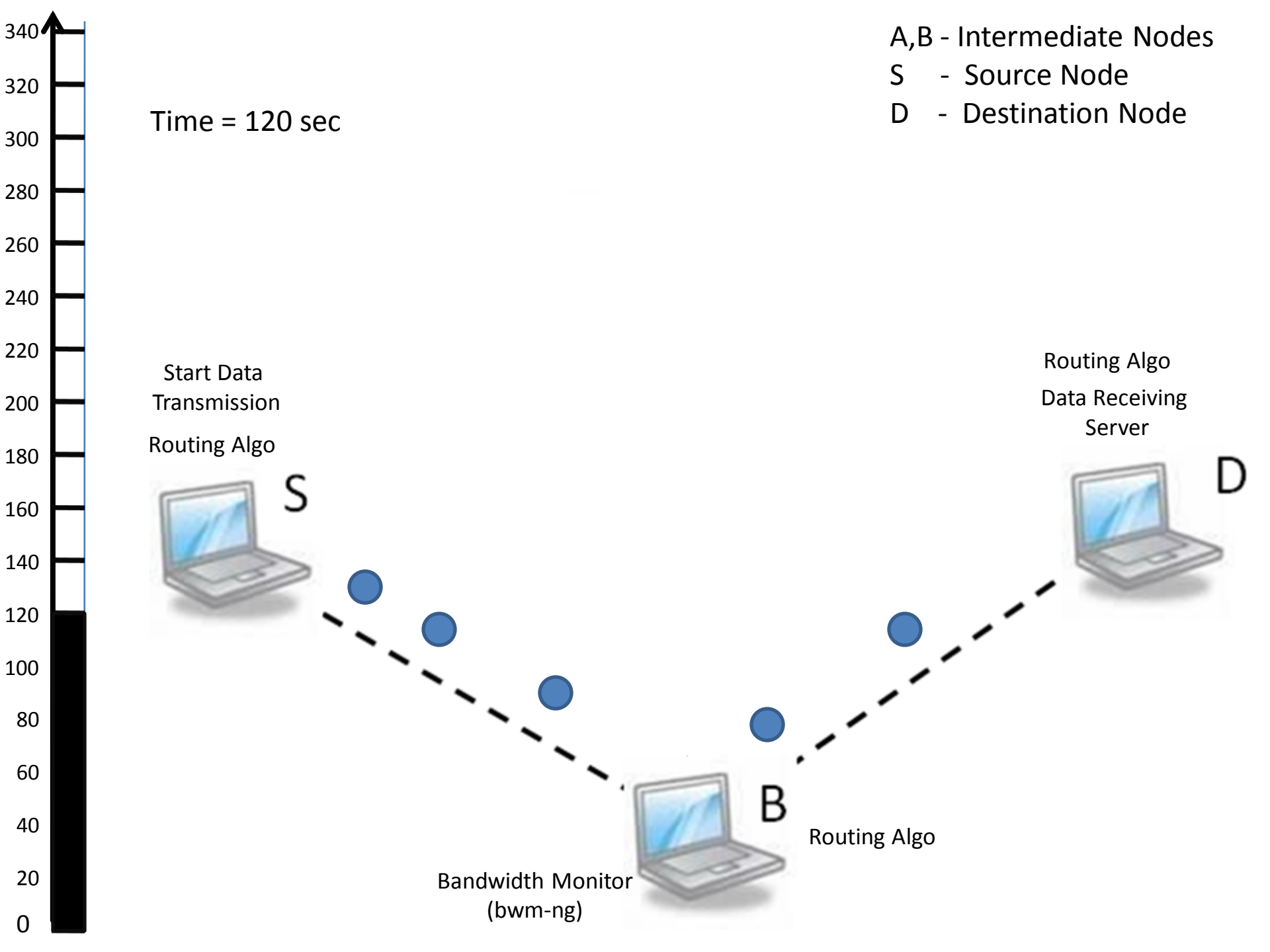
Testing Scenario

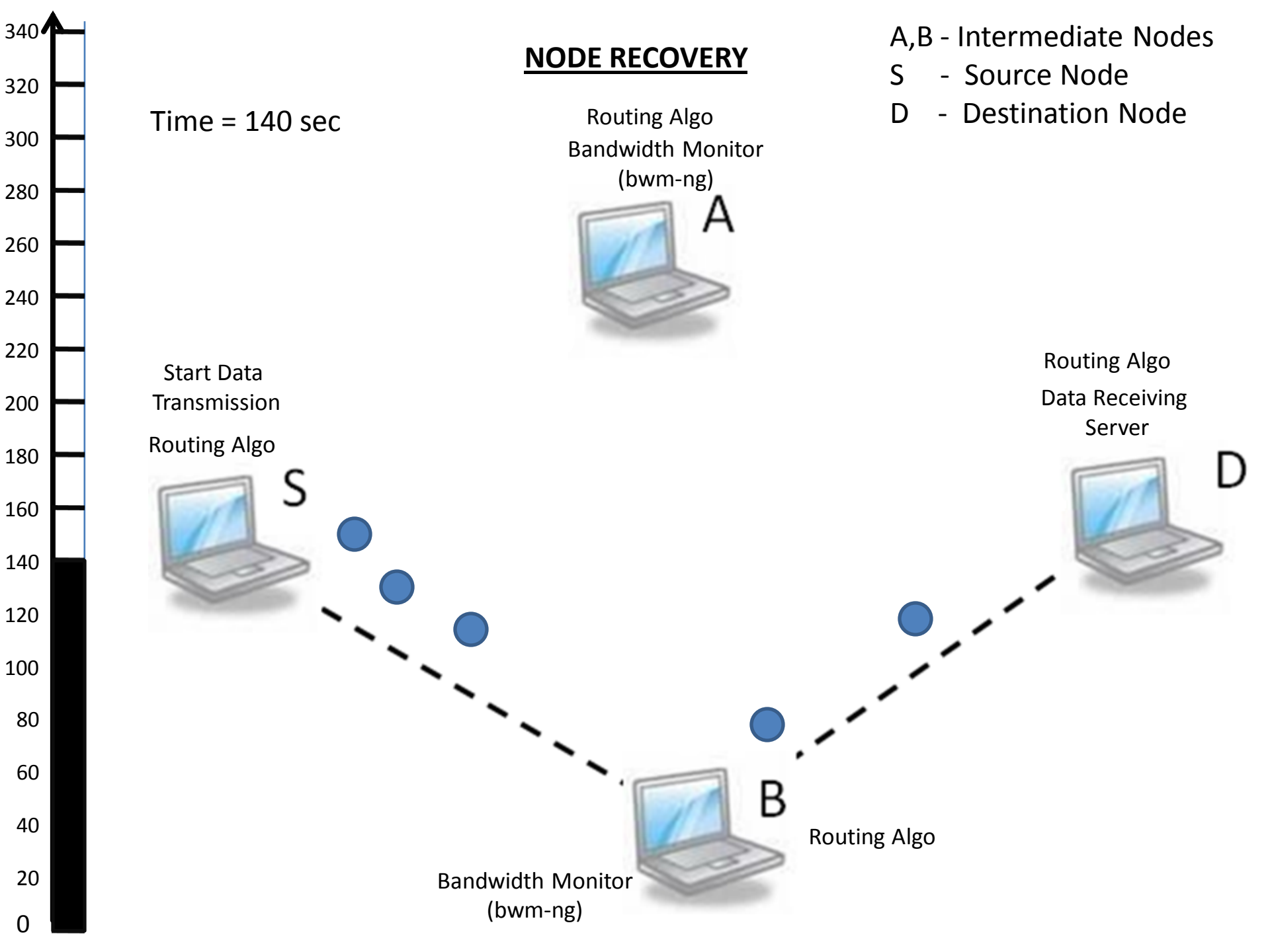

340
320
300
280
260
240
220
200
180
160
140
120
100
80
60
40
20
0
Time = 0 sec
A,B - Intermediate Nodes
S - Source Node
D - Destination Node
Bandwidth Monitor (bwm-ng)
A
Data Receiving Server
D
S
B
Bandwidth Monitor (bwm-ng)

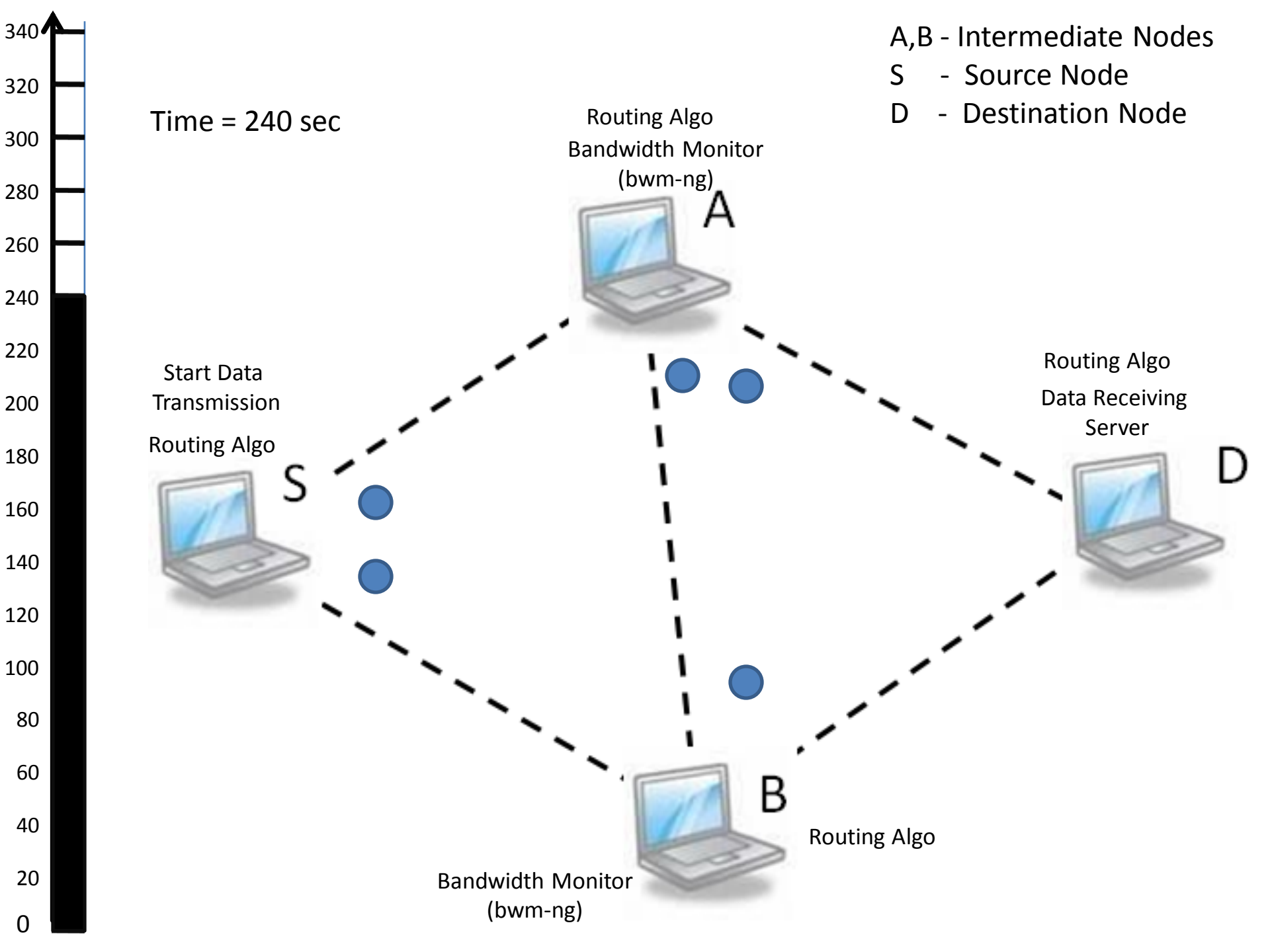

340
320
300
280
260
240
220
200
180
160
140
120
100
80
60
40
20
0
Time = 20 sec
A,B - Intermediate Nodes
S - Source Node
D - Destination Node
ROUTING ALGO
Bandwidth Monitor
(bwm-ng)
A
ROUTING ALGO
Data Receiving
Server
D
ROUTING ALGO
S
B
ROUTING ALGO
Bandwidth Monitor
(bwm-ng)

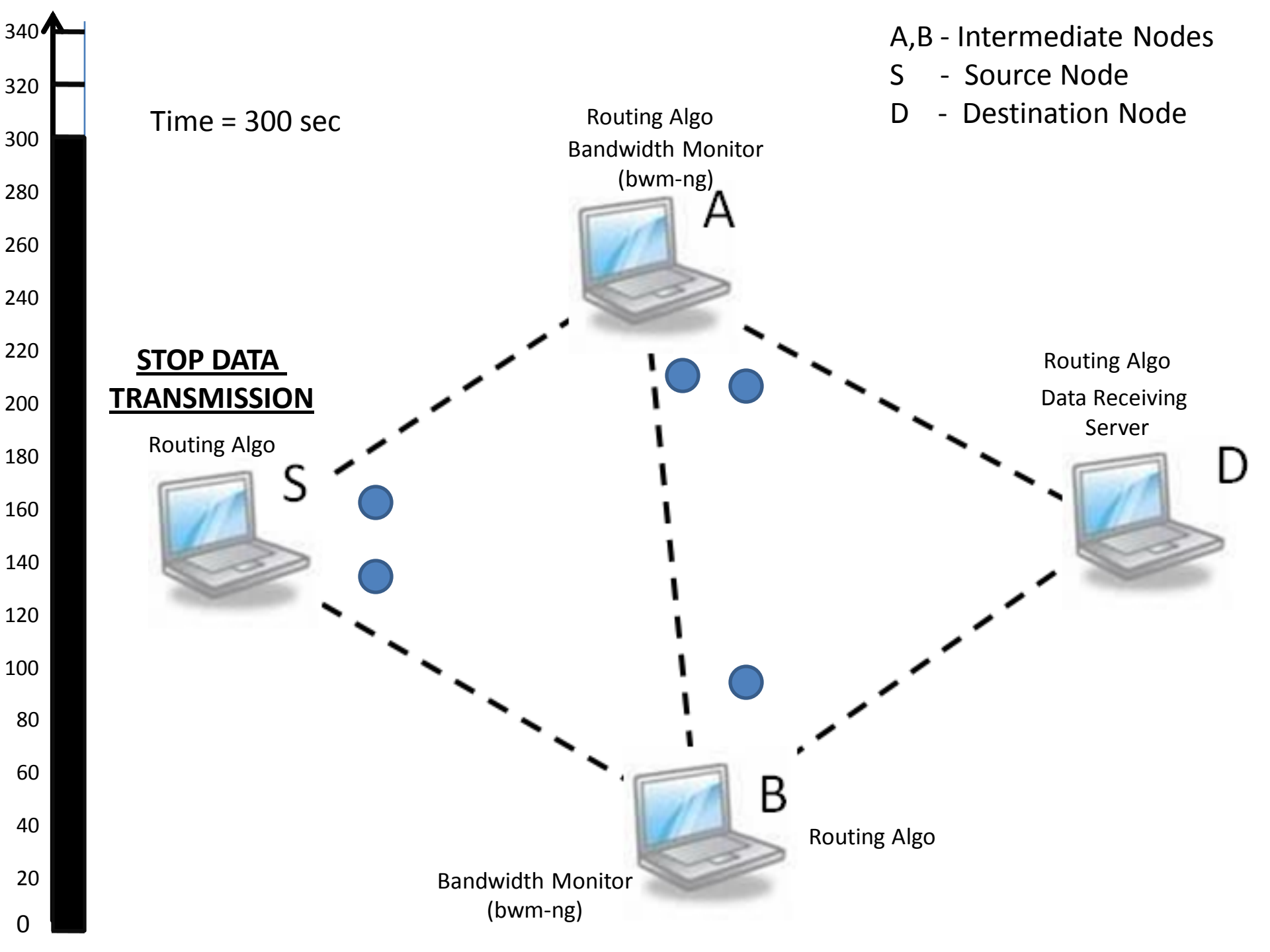

340
320
300
280
260
240
220
200
180
160
140
120
100
80
60
40
20
0
Time = 40 sec
A,B - Intermediate Nodes
S - Source Node
D - Destination Node
Routing Algo
Bandwidth Monitor
(bwm-ng)
A
START DATA
TRANSMISSION
Routing Algo
Data Receiving
Server
Routing Algo
S
D
B
Routing Algo
Bandwidth Monitor
(bwm-ng)

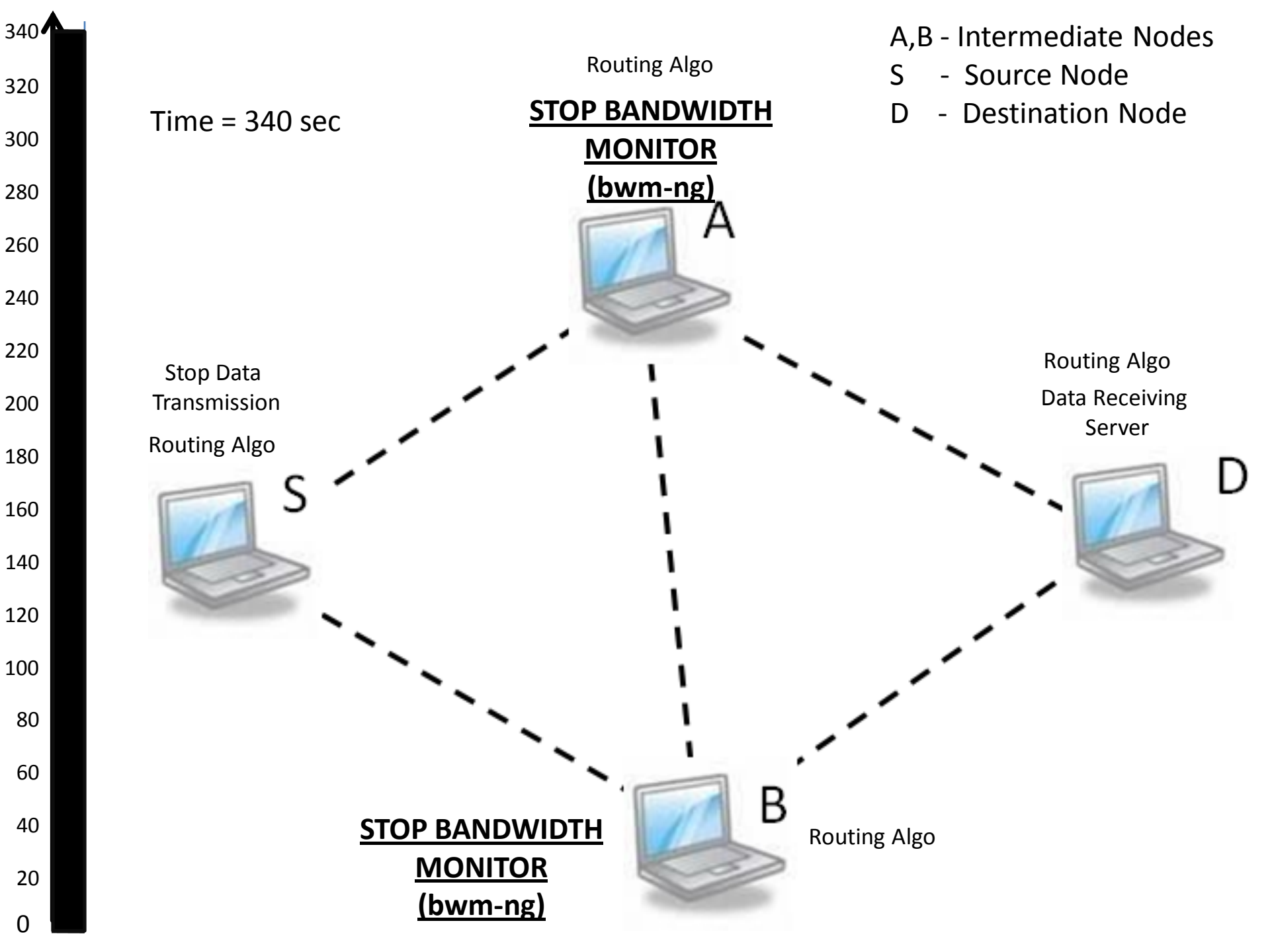

340
320
300
280
260
240
220
200
180
160
140
120
100
80
60
40
20
0
NODE FAILURE
A,B - Intermediate Nodes
S - Source Node
D - Destination Node
Time = 100 sec
Routing Algo
Bandwidth Monitor
(bwm-ng)
A
Start Data
Transmission
Routing Algo
S
Routing Algo
Data Receiving
Server
D
B
Routing Algo
Bandwidth Monitor
(bwm-ng)


340
320
300
280
260
240
220
200
180
160
140
120
100
80
60
40
20
0
Time = 120 sec
A,B - Intermediate Nodes
S - Source Node
D - Destination Node
Start Data Transmission
Routing Algo
S
Routing Algo
Data Receiving Server
D
B
Routing Algo
Bandwidth Monitor (bwm-ng)


340
320
300
280
260
240
220
200
180
160
140
120
100
80
60
40
20
0
NODE RECOVERY
A,B - Intermediate Nodes
S - Source Node
D - Destination Node
Time = 140 sec
Routing Algo
Bandwidth Monitor
(bwm-ng)
A
Start Data
Transmission
Routing Algo
S
Routing Algo
Data Receiving
Server
D
B
Routing Algo
Bandwidth Monitor
(bwm-ng)

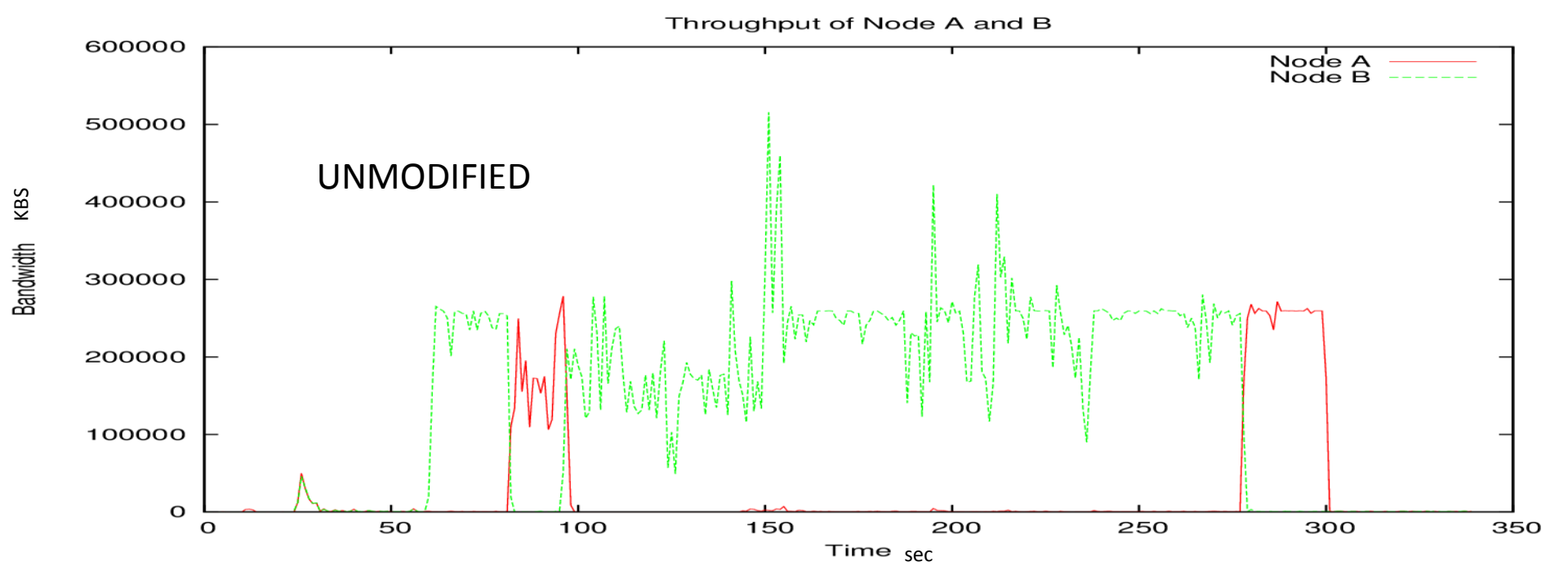

340
320
300
280
260
240
220
200
180
160
140
120
100
80
60
40
20
0
Time = 240 sec
A,B - Intermediate Nodes
S - Source Node
D - Destination Node
Routing Algo
Bandwidth Monitor
(bwm-ng)
A
Start Data
Transmission
Routing Algo
S
Routing Algo
Data Receiving
Server
D
B
Routing Algo
Bandwidth Monitor
(bwm-ng)

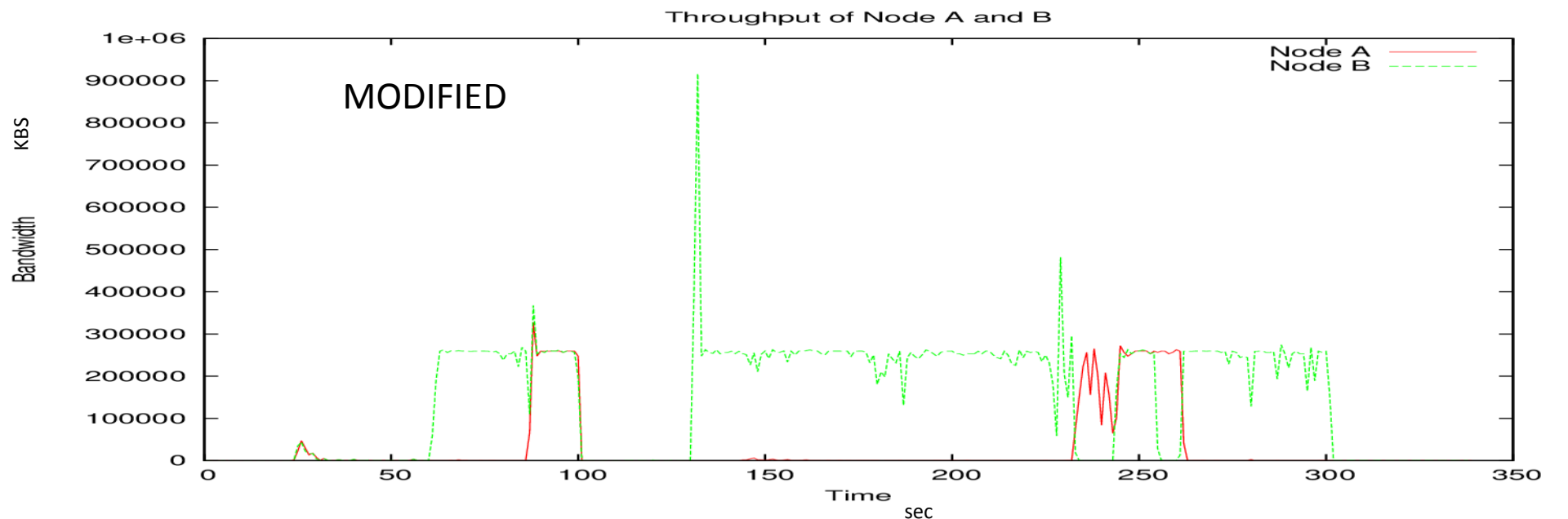

340
320
300
280
260
240
220
200
180
160
140
120
100
80
60
40
20
0
Time = 300 sec
A,B - Intermediate Nodes
S - Source Node
D - Destination Node
Routing Algo
Bandwidth Monitor
(bwm-ng)
A
STOP DATA
TRANSMISSION
Routing Algo
S
Routing Algo
Data Receiving
Server
D
B
Routing Algo
Bandwidth Monitor
(bwm-ng)

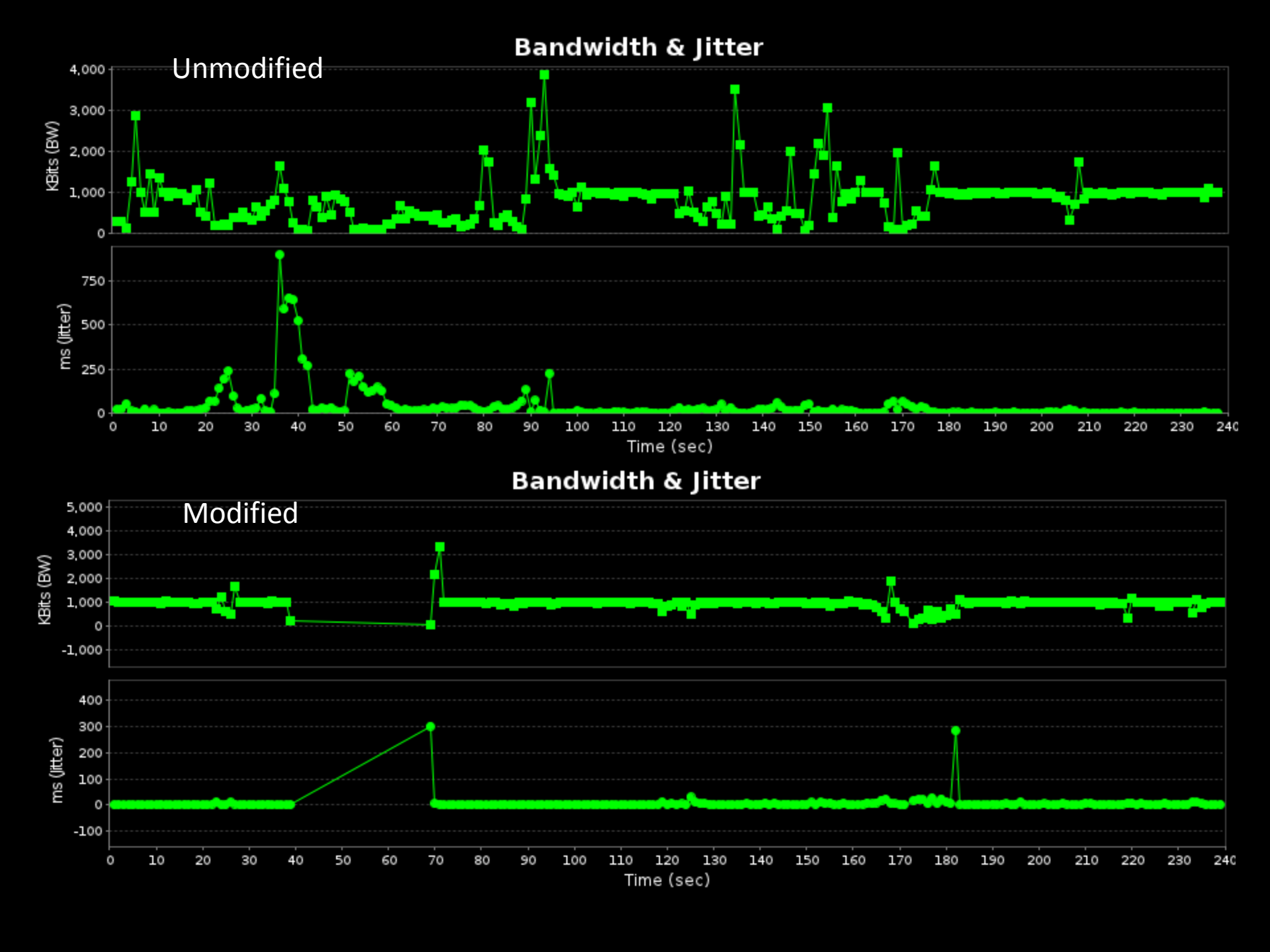

340
320
300
280
260
240
220
200
180
160
140
120
100
80
60
40
20
0
A,B - Intermediate Nodes
S - Source Node
D - Destination Node
Time = 340 sec
Routing Algo
STOP BANDWIDTH MONITOR (bwm-ng)
A
Stop Data Transmission
Routing Algo
S
Routing Algo
Data Receiving Server
D
STOP BANDWIDTH MONITOR (bwm-ng)
B
Routing Algo

Comparison 1

Unmodified vs. Modified Protocol


Throughput of Node A and B
UNMODIFIED
Node A
Node B
Bandwidth KBS
Time sec
600000
500000
400000
300000
200000
100000
0
0
50
100
150
200
250
300
350


Throughput of Node A and B
MODIFIED
Node A
Node B
Bandwidth KBS
Time sec
1e+06
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
0
50
100
150
200
250
300
350

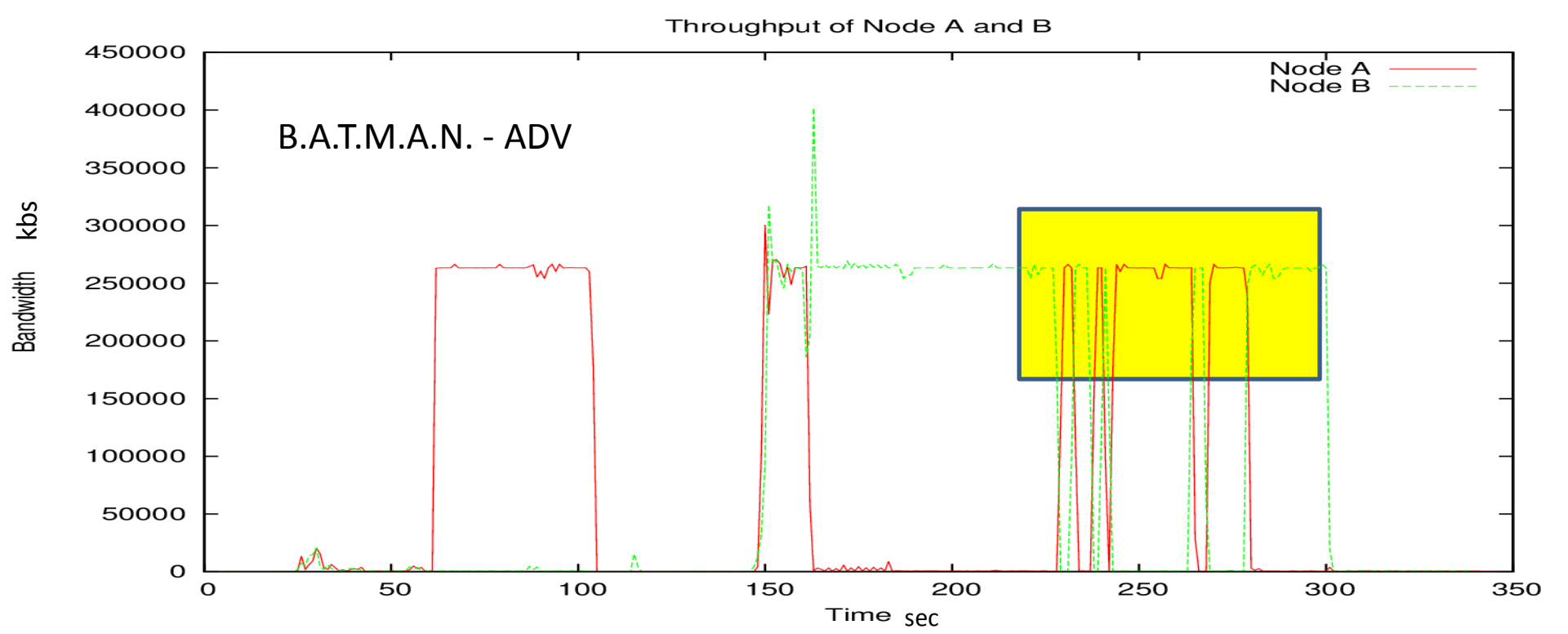

Unmodified
Bandwidth & Jitter
KBits (BW)
4,000
3,000
2,000
1,000
0
ms (jitter)
750
500
250
0
0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240
Time (sec)
Modified
Bandwidth & Jitter
KBits (BW)
5,000
4,000
3,000
2,000
1,000
0
-1,000
ms (jitter)
400
300
200
100
0
-100
0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240
Time (sec)

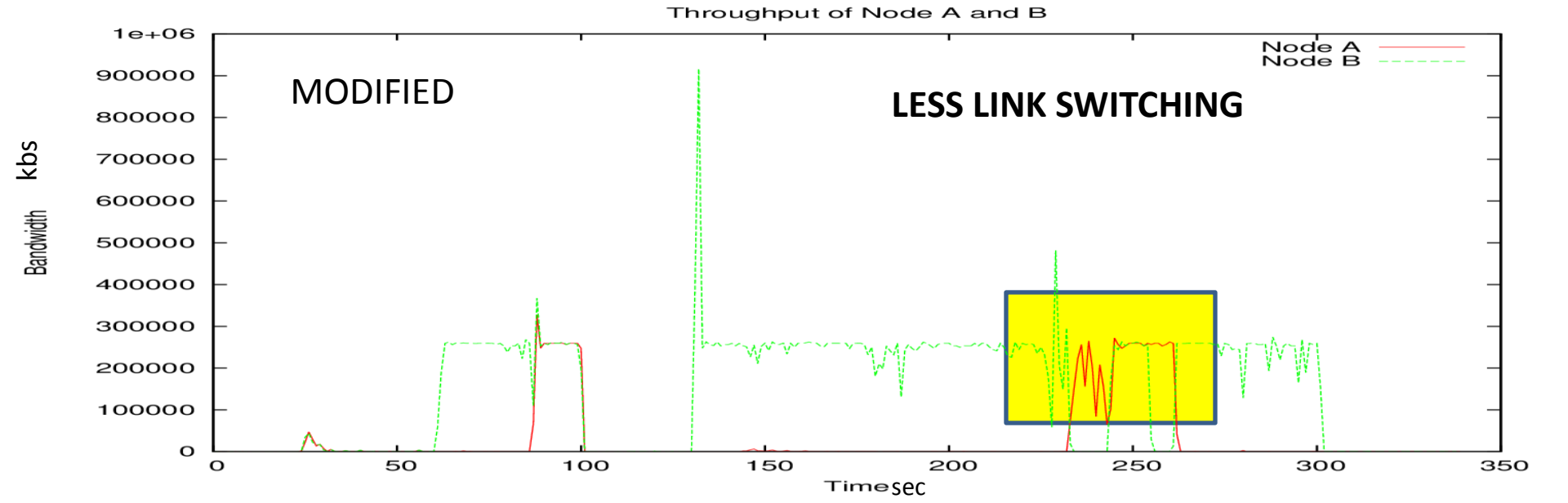
Results

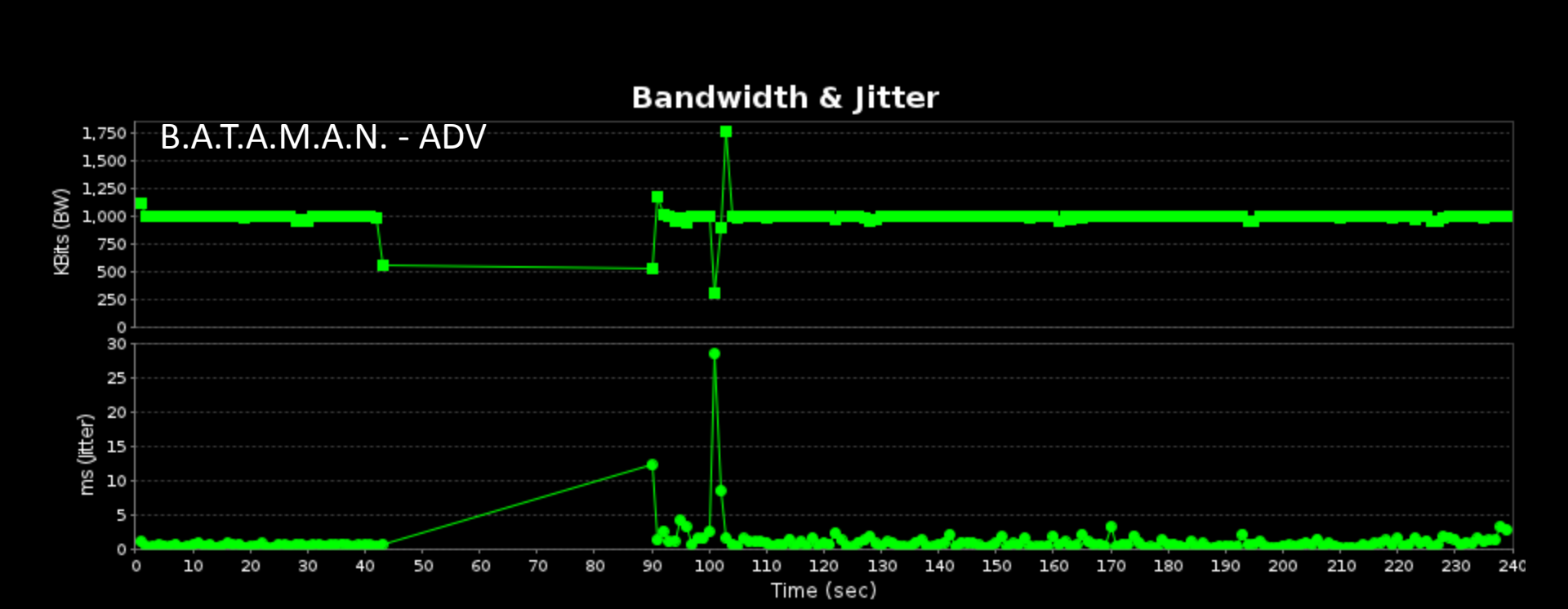
- Reduction in bursty data
- Reduction in Jitter
- Reduction in Packet Loss from 16% to 11%
- Reduction in out-of-sequence packets

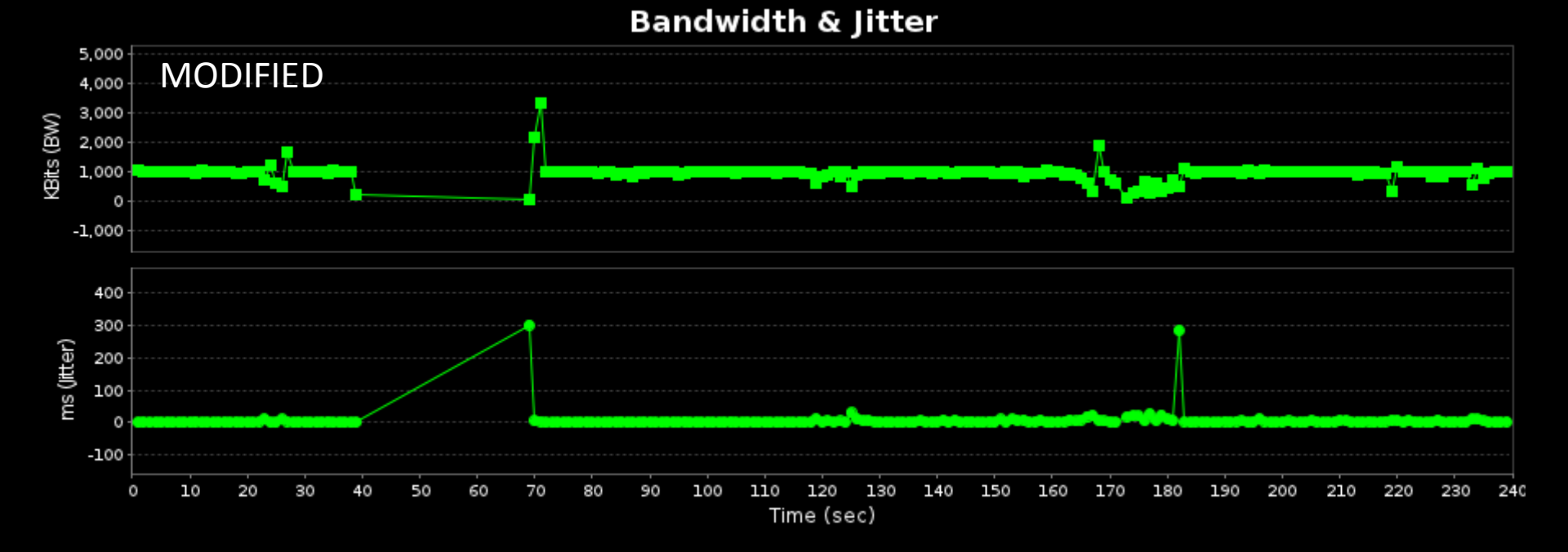
Comparison 2

Modified Protocol vs. B.A.T.M.A.N.-ADV


Throughput of Node A and B
Node A
Node B
B.A.T.M.A.N. - ADV
Bandwidth kbs
Time sec
0
50000
100000
150000
200000
250000
300000
350000
400000
450000
0
50
100
150
200
250
300
350


Throughput of Node A and B
Node A
Node B
MODIFIED
LESS LINK SWITCHING
Bandwidth kbs
Time sec
0
100000
200000
300000
400000
500000
600000
700000
800000
900000
1e+06
0
50
100
150
200
250
300
350


Bandwidth & Jitter
B.A.T.A.M.A.N. - ADV
KBits (BW)
1,750
1,500
1,250
1,000
750
500
250
0
ms (jitter)
30
25
20
15
10
5
0
0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240
Time (sec)

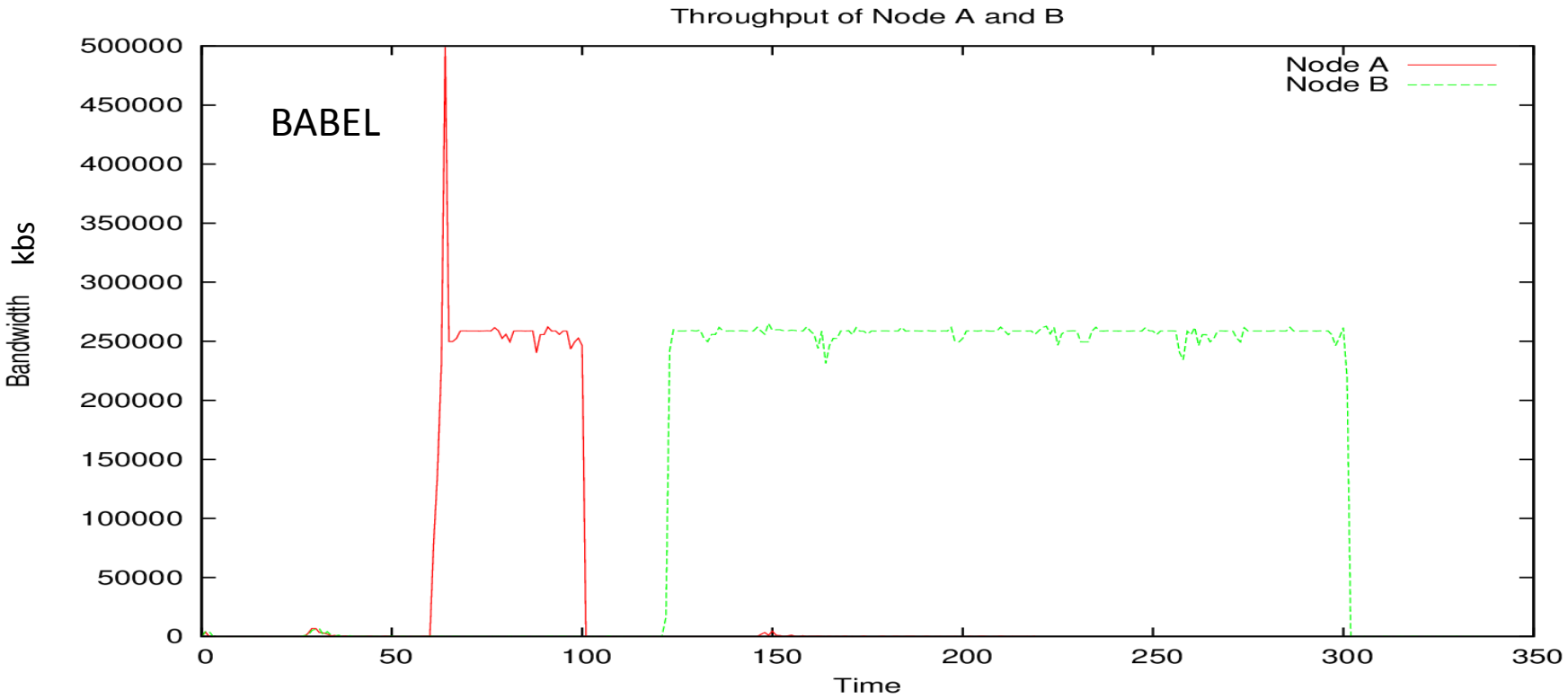

Bandwidth & Jitter
MODIFIED
KBits (BW)
5,000
4,000
3,000
2,000
1,000
0
-1,000
ms (jitter)
400
300
200
100
0
-100
0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240
Time (sec)

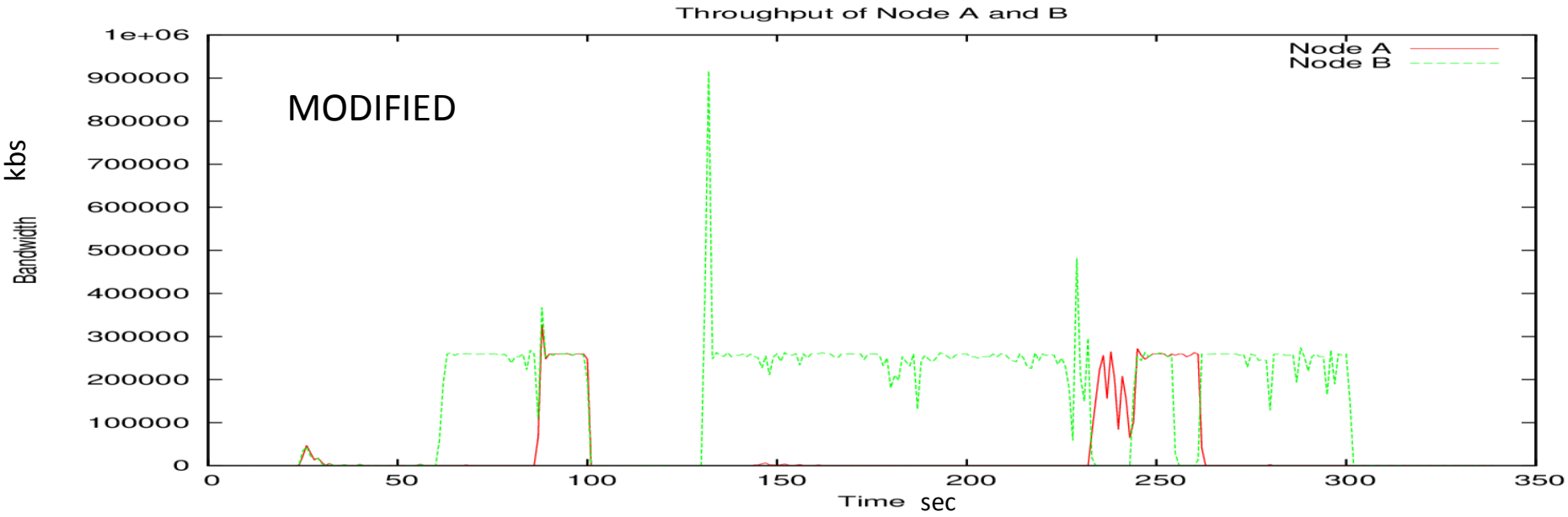
Results

- Instability after route repair in B.A.T.M.A.N.
- Bandwidth drops to Zero at node failure in B.A.T.M.A.N.-ADV
- Jitter is less in B.A.T.M.A.N. –ADV.

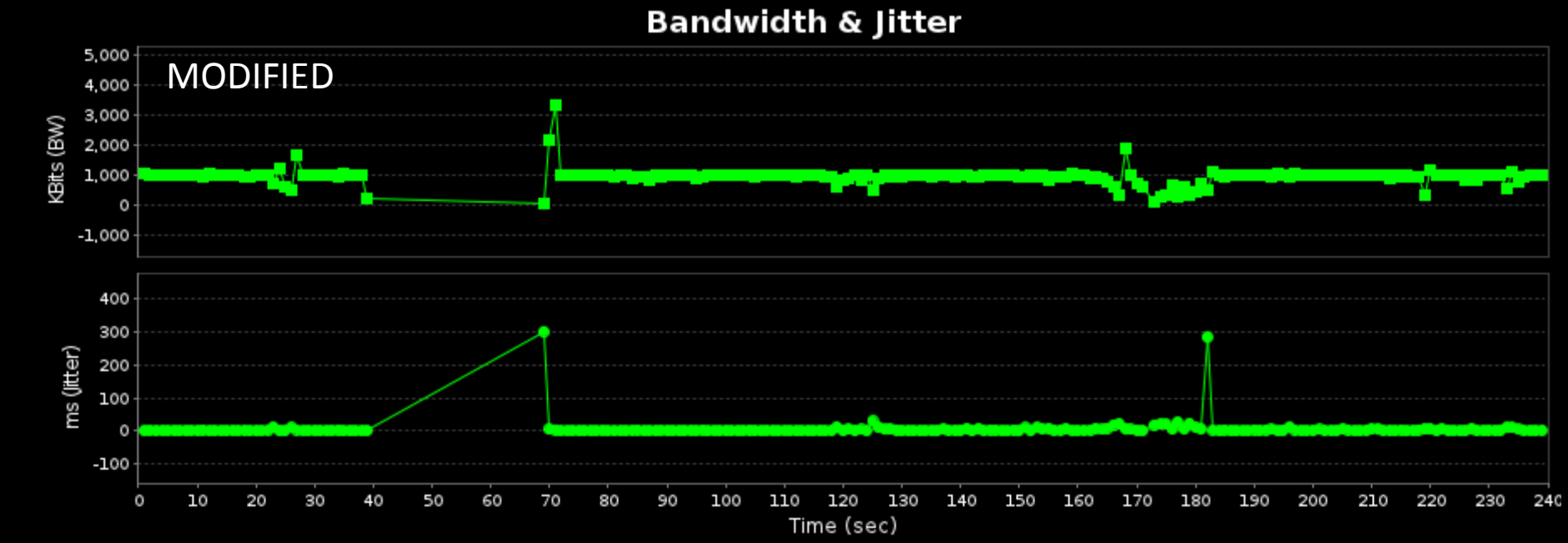
Comparison 3

Modified vs. BABEL


Throughput of Node A and B
BABEL
Node A
Node B
Bandwidth kbs
500000
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
0
50
100
150
200
250
300
350
Time


Throughput of Node A and B
MODIFIED
Node A
Node B
Bandwidth kbs
1e+06
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
0
50
100
150
200
250
300
350
Time sec


Bandwidth & Jitter
BABEL
KBits (BW)
2,500
2,000
1,500
1,000
500
0
ms (jitter)
30
25
20
15
10
5
0
0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240
Time (sec)


Bandwidth & Jitter
MODIFIED
KBits (BW)
5,000
4,000
3,000
2,000
1,000
0
-1,000
ms (jitter)
400
300
200
100
0
-100
0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240
Time (sec)

Results

- Bandwidth drops to zero at node failure in Babel.
- Babel is more stable

Conclusion

- Visible improvements are observed over existing approaches but more work is required to make the protocol suitable for real world applications

Future Scope

- To resolve physical and logical hop problem in Babel and B.A.T.A.M.A.N-ADV.
- Reduce overhead of modified protocol.
- Metric tweaks to improve performance.

Demo